UNCLASSIFIED

AD NUMBER

AD484444

NEW LIMITATION CHANGE

TO

Approved for public release, distribution unlimited

FROM

Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; Feb 1966. Other requests shall be referred to Rome Air Development Center, Griffiss AFB, NY 13440.

AUTHORITY

RADC USAF ltr, 17 Sep 1971

THIS PAGE IS UNCLASSIFIED

RADC-TR-66-143
Final Report

ADVANCED COMPUTER ORGANIZATION

F. T. Baker
W. E. Triest
et. al.
International Business Machines

TECHNICAL REPORT NO. RADC-TR-66-143

May 1966

ADVANCED COMPUTER ORGANIZATION

F. T..Baker
W. E. Triest
International Business Machines

AFLC, GAFB, N.Y., [illegible] June [illegible]

FOREWORD

This report is submitted as a result of performance on Contract AF 30(602)-3573 awarded by the Rome Air Development Center, Air Force System Command, Griffiss AFB, New York to the Federal Systems Division, IBM Corporation, 7220 Wisconsin Avenue, Bethesda, Maryland. This study, leading to the development of an advanced general-purpose computer organization featuring content addressing and parallel processing of data, was performed during the period November 5, 1964 to January 4, 1966 and was carried out by F. T. Baker, C. H. Forbes, N. Jacobs, J. D. Schenken, W. E. Triest, and T. P. Walker, Jr. The authors wish to express their gratitude to the RADC project engineer, F. A. Dion, and also to R. J. Ferris of RADC for their cooperation and suggestions throughout the project. Valuable consultation was provided by F. A. Behnke, A. R. Geiger, J. E. Griffith, A. B. Lindquist, H. E. Peterson, J. H. Pomerene and R. R. Seeber of the IBM Corporation.

RADC Project Number is 4594; Task Number 459406.

This report has been reviewed and is approved.

Approved: FRANK J. TOMAINI
Chief, Info Processing Branch

Approved: ROBERT J. QUINN, JR., COLONEL, USAF
Chief, Intel and Info Processing Div.

FOR THE COMMANDER: IRVING J. GABELMAN
Chief, Advanced Studies Group

ABSTRACT

This study resulted in a design of an advanced general-purpose computer, including its functional organizational and programming. The design is based on content-addressable, parallel search memories and the computer has parallel processing capability. It resulted from investigations in several important areas of non-numeric processing and symbol manipulation, and the design studies which were carried out in each area. In addition to the general-purpose computer and the individual design studies, a number of associative processing techniques were developed for use with such equipment.

CONTENTS

CONTENTS (Cont'd)

ILLUSTRATIONS

LIST OF TABLES

LIST OF TABLES (Cont'd)

Section I

INTRODUCTION

This study was undertaken primarily to investigate the applicability of associative computer organizations to problems involving non-numeric processing. Previous work in automatic data extraction, automatic indexing, and formatted file processing suggested that the techniques of content addressing and parallel processing afforded by an associative processor would offer significant advanta[ges] both in performance and in ease of programming for such applications. The ove[r]all goal therefore was to design an associative processor useful for such work, while at the same time retaining the ability to perform numeric processing when required.

The design approach was to work from the specific to the general. Initially problem areas characteristic of those commonly encountered in non-numeric processing were identified. After examining a number of non-numeric problems five areas were selected as representative. In general, these were problems characteristic of those encountered in systems designed for automatic input (e.g., indexing and data extraction) from narrative documents and the storage, processing, and retrieval of a large volume of information from and about these documents. The five areas were:

1. Text searching and term identification (automatic data extraction).
2. Generation of text statistics

3. Text processing using a large dictionary

4. Formatted file query

5. Pattern classification

After the selection of these problem areas for study a special purpose processor was independently designed for each problem area. The performance of each processor was then compared to that of a conventional processor on the basis of a specific problem characteristic of that area. A description of each of these areas and the processors designed for them is contained in Sections II - VI. Once designs had been completed in all areas, features were identified which were useful in several problem areas. Finally, these features were incorporated into a general-purpose associative processor.

Concurrent with the design of these five processors, supplementary studies were conducted to examine three additional topics which it was felt would contribute to design of the general processor. Text encountered as input to a computer frequently has a high percentage of errors resulting from external data handling and processes such as long-distance data transmission or optical reading. A study was therefore made to determine a measure which would be useful in an associative processor in determining the "closest" word to an erroneous word encountered in text. Current computers frequently have operation sets which are "incomplete" in the sense that potentially useful operations are missing or in which cumbersome operations could be replaced by simpler ones. In order to avoid these faults, a study of methods in which operation sets could be structured was performed. Finally, some frequently performed types

of numeric processing were examined to see what techniques could be applied to their solution on associative processors. These studies are described in Section VII.

These efforts led to the development of the General Purpose Associative Processor (GPAP) described in Section VIII. The value of the approach was confirmed by the identification in the individual problem area studies of some unique features which were valuable over a broad range of processing. These were:

1. Capability for "linking" consecutive registers of an associative storage unit so that a match on the contents of one register can be indicated in the immediately preceding or following register, thus allowing for simple handling of important structural aspects of data.

2. High-speed parallel searching, input and output of bulk files of data by use of a "channel processor" containing two associative search matrix units, thus allowing for the performance of complex logical queries external to the central processor with resulting economy of data transmitted and central processor time wasted.

3. The use of semi-autonomous associative processing units of varying speed, capacity and cost to permit increased overall system performance.

4. Flexible handling of bits, characters, or words with consequent ability to process many types of data.

The resulting design incorporates all of these features in addition to some interesting architectural characteristics not found in the individual processors.

The end result of this effort, then, is the functional organization of an advanced general-purpose computer featuring content-addressing and parallel searching capabilities. Detailed logical design can be carried out based on the organization specified. Indeed, in some areas of unusual novelty or complexity, detailed logical design was carried out where necessary to ensure feasibility. (It is important to note, however, that where this was done, feasibility and not optimality was the only object). Section IX contains an evaluation of the resulting design and its potential.

Design of digital computers has moved from the original concepts of von Neumann, et al, by means of hardware inventions, improved components, larger memories, programming languages and techniques, and new and important application areas involving non numeric processing. Non-numeric processing is hardly hinted at in the Princeton report[1], and, at the time of its writing, the major manufacturers of machines for business data handling were not aware of the potential extensions of digital computers into areas of non numerical problems. As such extensions were developed, clever programming and special compilers were developed, and extensive research areas were opened up for investigations into the basic nature of non-numeric problems. The relevance of thi[illegible] design of an advanced machine organization, including associative memories and parallel processing capability, is the implication that one cannot realistically evaluate the efficacy of such an advanced machine without anticipating a succession of innovations in machine application techniques.

---

1. Burks, A. W., Goldstine, H. H., and von Neumann, J., "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument." Institute for Advanced Study, Princeton, N.J., 1946

Thus, in addition to the final design, and perhaps even more important for future designers and users of associative processors, the techniques and features developed in the individual problem area studies promise to provide guidelines in this general area. The problems selected are practical, and in some cases the methods studied are applicable on a variety of computer organizations. Thus, while the proposed design could be implemented within a five-year time period, many of the techniques are currently useful on present-day processors, both conventional and associative.

Section II

DATA EXTRACTION

PROBLEM STATEMENT

The data extraction problem is the process of extracting information of interest to the user from textual data. The user specifies the type and character of data desired, and the extraction processor performs the process to yield the desired result.

The type and character of the problem is stated in the form of character string configurations. The configurations may be limited in nature, i.e., exactly matching the text character by character (e.g., "F-111"). The configuration statement may be such that certain classes of characters fall into some general form (e.g., "B-dd", where "d" may represent any numeric digit). Or, the statement may consider this type of configuration and further specify some particular concatenation of these types. Variable position of the parameters within the concatenation may also be supplied. The variations available to the user are, in effect, limitless when viewed from the processor standpoint, and the system required to process this type of input to achieve the desired result will be highly complex.[1]

---

[1] ANEXOR, COMIT, SNOBOL, LISP 1.5, etc., are examples of programming languages used to aid in the solution of data extraction.

CHARACTERISTICS OF THE PROBLEM

The data extraction problem covers a wide range of data sources. These might include newspapers, periodicals, financial journals, intelligence reports, books, etc. The data format in these documents varies widely from unformatted text (books) to semiformatted text (financial journals,[1] intelligence reports in unfinished form) to completely formatted reports finished intelligence reports, baseball boxscores). The data to be extracted from these sources also shows wide variation, such as in the earnings of a corporation or the names of officers of the company. The data can also be the location of some activity being reported on in the news article or intelligence report, or the itenerary of some VIP whose movements are the subject of the intelligence report in question.

In any of these examples, the prime consideration that relates each example to the same general problem is the structure of the material being considered. Document lengths and formats may change drastically from document to document, and within the same class of documents. Vocabulary will certainly change. But, for each class of document, the underlying structure will remain virtually unmoved and unchanged. This structure will enable the analyst to determine the algorithms necessary for extracting data.

---

1 "Automatic Information Extraction", C.H. Forbes, 31 May 1962, AIDS/SAC Subsystem Working Paper.

Structure is the prime characteristic of the data which remains relatively invariant. It provides the major clue as to how the data is to be extracted from the document. Document structures may take the form of hierarchies or groupings, such as pattern analysis, maze solviving, or processing of directed graphs. In text, structure is a reflection of the language used to describe the information being presented. In this way we notice that names of persons tend to appear in specified manners. That is, the appearance of "b̸ Mr. b̸" in text is a flag that signifies that the next characters form the name of someone. In like manner, other structures yield clues to the data for all classes of reports and documents.

The value of structure can most graphically be illustrated by the following 'gedenken experiment'. Consider a document and postulate that the words of that document are to be rearranged in alphabetical order (or in word length order or in some other arbitrary order depending on individual characteristics of the words and not related to their position in text). All of the words will still be there, but meaning will be lost and most of the information of the document will be irretrievably destroyed. The structural relationships between words will have been altered (replaced by a new structure) so that little indication remains of the original structure. This transformation will effectively prevent the efficient utilization of the document as an information source.

As an example of data that might be useful if extracted, consider the following request for information from a class of reports: "Identify those documents containing statements by VIPs." The documents under consideration are reports of various activities and include reports of public statements made by VIPs as well as a wide range of other activities ascribed to them.

An analysis of a subset of the documents used revealed that names of VIPs appeared in a restricted environment in the text and that the 'statement' documents had such clue words as speech, statement by, said, quote. 'Nonstatement' documents were found to have different words and phrases than the above in the areas related to the supposed VIP names. Thus it is possible to discriminate between the VIP statement documents and the VIP nonstatement documents. This also holds true for the general activities documents which contain no VIP information.

A second example in this class is the analysis of financial journal reports of earnings of companies and corporations. This data is contained in narrative reports that discuss the per share earnings as well as the overall earnings of the company. Earnings are discussed in terms of net after taxes, gross earnings, etc. For example,[2] company earnings are sometimes expressed as a total value which is followed by the per share equivalent. The per share equivalent is sometimes expressed as A SHARE, PER SHARE, A COMMON SHARE, A PREFERRED SHARE. Since character strings are being described, it is necessary to account for the possible presence of the word COMMON or the word PREFERRED. In locating the desired information, variations of this kind need to be considered by means of the conditional statements that indicate acceptability of either the presence or the absence of the word in question.

---

[1] Memorandum to A. R. Geiger from J. D. Schenken, Private Communication, July 1963.

[2] ANEXOR, C.H. Forbes, 31 Jan 1963, IBM Working Paper.

That is, COMMON or PREFERRED in the data string is acceptable and the absence of them is also acceptable provided that no other term is in their place. It is often necessary to concatenate the terms of the search in a Boolean expansions of terms. That is, COMPANY, CORPORATION or INCORPORATED will be acceptable in the specified search location. Or, EAST, WEST, NORTH, or SOUTH not immediately preceded by THE is acceptable.

A specific example of these applications is the parameters specifying sales, earnings, etc, expressed as total or per share. Character string "A" is defined as either

1. EARNED or SALES or EARNINGS or INCOME or NET or PROFIT followed by:

2. OF or TOTALED or WAS or WERE or INCREASED TO or ROSE TO or DECLINED TO or FELL TO or SLIPPED TO

Character string "B" is defined as some variation of the basic pattern "$ total value, equivalent share price, FROM, $ total value, equivalent share price." Therefore,

1. For share prices search for "$" followed by numeric digits, or

2. Two digits followed by CENTS.

Either or both of the share price portions of the string "B" may be legitimately missing. FROM and all of the subsequent data may also be missing. The total value may be expressed as all-numeric or some alphabetic representation of a numeral, i.e., ONE, TWO, The data to be output from this search is the string "A" from text followed by the string "B."

The foregoing have been relatively restricted to types of problems in which configuration has played a major role in the determination of the data to be

extracted. This is just one of the areas of data specification that must be treated for this problem. The following is a general description of configuration as related to structure for data extraction.

Configuration. Configuration means the characters which will appear in the data string, and is essentially a description of the data string. This description may be in the form of an explicit description as seen in the foregoing examples or in the form of an implicit description, as in the case of the share price above.

Explicit Description. For data strings which exhibit well known characteristics such that the description is exact and unique, the Explicit Description is used. It specified each data character and relation to other characters in the string, i.e., EARNED, EARNINGS,

Implicit Description. For those cases where there exists no rigid definition of the data string in the sense of the Explicit Description, or where such a description is not possible due to the extensive lists involved, the Implicit Description is utilized. Such things as any number, where number is suitable defined to the program, any three letters followed by 2, 3, or 4 special characters fall into the category of Implicit Description. A number might be any string of two or more consecutive digits bounded on each end by one or more blanks. Imbeded commas would be allowed if they divide the digits into groups of three digits, except for the leading digits where one or two digits would be an acceptable alternative. Imbedded periods (decimal point) would be acceptable if only one exists and it is adjacent on the right to at least one digit. Special characters

would be defined in a list which would contain but not be limited to:

"#", "$", "%", "&", "(", ")".

Other classes of characters might be:

1. Letters, A, B, C, D, ...
2. Control characters, line feed, carriage return bell, upper case shift, lower case shift ....
3. Capitalized letters
4. Illegal or other undefined characters.

Another portion of the structure available for analysis is Position, which is the relationship of the particular string to other strings in the data set. It may be specified in terms of rank, sequence, and area.

Rank. Rank refers to the first, second, tenth, etc., string of the sample; the first, etc., occurrence of some particular string in the sample; or the nth string (work, character, etc.) before or after some other specified string in the data. For example, the first date-time group in a report may indicate the date of the activity while the second might be the date of the report. The last such group could be the date that the report was received for processing. Since all of the date-time groups possess the same or similar characteristics, only their rank discriminates them in the absence of field identifiers.

Sequence. This property relates to the order of appearance in the data string. If string "A" is specified to appear before string "B" then "B" before "A" is not acceptable. If, in searching for the data in a document, the sequence IN 1961, JUNE WENT TO PARIS appears, then the string JUNE and the string 1961 will not be recognized as the date, since it is required (at least for this example) that the name of the month precede the number of the year. Sequence may also

include the factor of immediacy in text. That is, one specified string must be found immediately adjacent, either before or after, the other string under consideration. As a similar example, consider the rearrangement of the preceding case. JUNE WENT TO PARIS IN 1961 would not have JUNE 1961 identified as the date since the strings are not immediately adjacent.

Area. Area indicates that the character string must be located in some specified position or area within the overall string. Location may be specified in terms of document content. Most documents contain such elements as paragraphs, sections, sentences and other physical characteristics definable in terms of character counts, special control characters, and etc. To limit search time, or preclude "false drops" in the extraction process, the search may be restricted to some particular area of the document. Consider a report concerned with friendly activities in one section and enemy activities in another. The terms and constructions in both sections will be quite similar so that the most significant delimiter for separation of the two types of activity information is the section of the report in which it appears.

Area in reports can be specified in terms of starting points, a starting point and a direction or distance, or a starting point and an ending point. The starting point may be sepcified in terms of number of characters from the beginning of the document, number of sentences from some fixed point of the text, the beginning of some section or paragraph. It may be specified in terms of some particular string of characters such as JOHN Q. JONES. Direction may be given as forward or backward—toward the beginning of the document or the end. Distance may be given by a specific number of characters or words. It may be specified as the number of sentences or paragraphs.

PROCESSOR DESIGN

The basic processor organization is divided into three functional memory and processing units (see Figure 1): input editing processor, general search processor, and output formatting processor. The units are conceived as separate in this application. However, depending upon the particular parameters, the second and the third sections of the processor might be combined into one unit and share the functions.

The internal organization of the three memories is similar and consists of memory registers organized to contain one character of data each, with several additional bits for control information and marker bits for use by the associative hardware in marking the various results of associative operations on that register. The memory/processor units have multiwrite capability, allowing for simultaneous modification of the contents of several registers of the memory.

The memory units are organizable under program control to allow for two modes of operation (Figure 2). The first is the character mode (CM) and sets the memory logic for one character per storage position. All associative operations, as well as read and write operations, are available during this mode of operation. The second mode, register mode (RM), allows the programmer to select from 2, 4, or 8 characters per word in the memory. The memory is then organized so that references to it select groups of 2, 4 or 8 of the single-character registers at a time. RM is used for data transmission both to and from the memory as well as within the memory unit.

Associative operations of the memory/processor unit are not available in this mode. Only those operations used for data transfer are now allowed. The

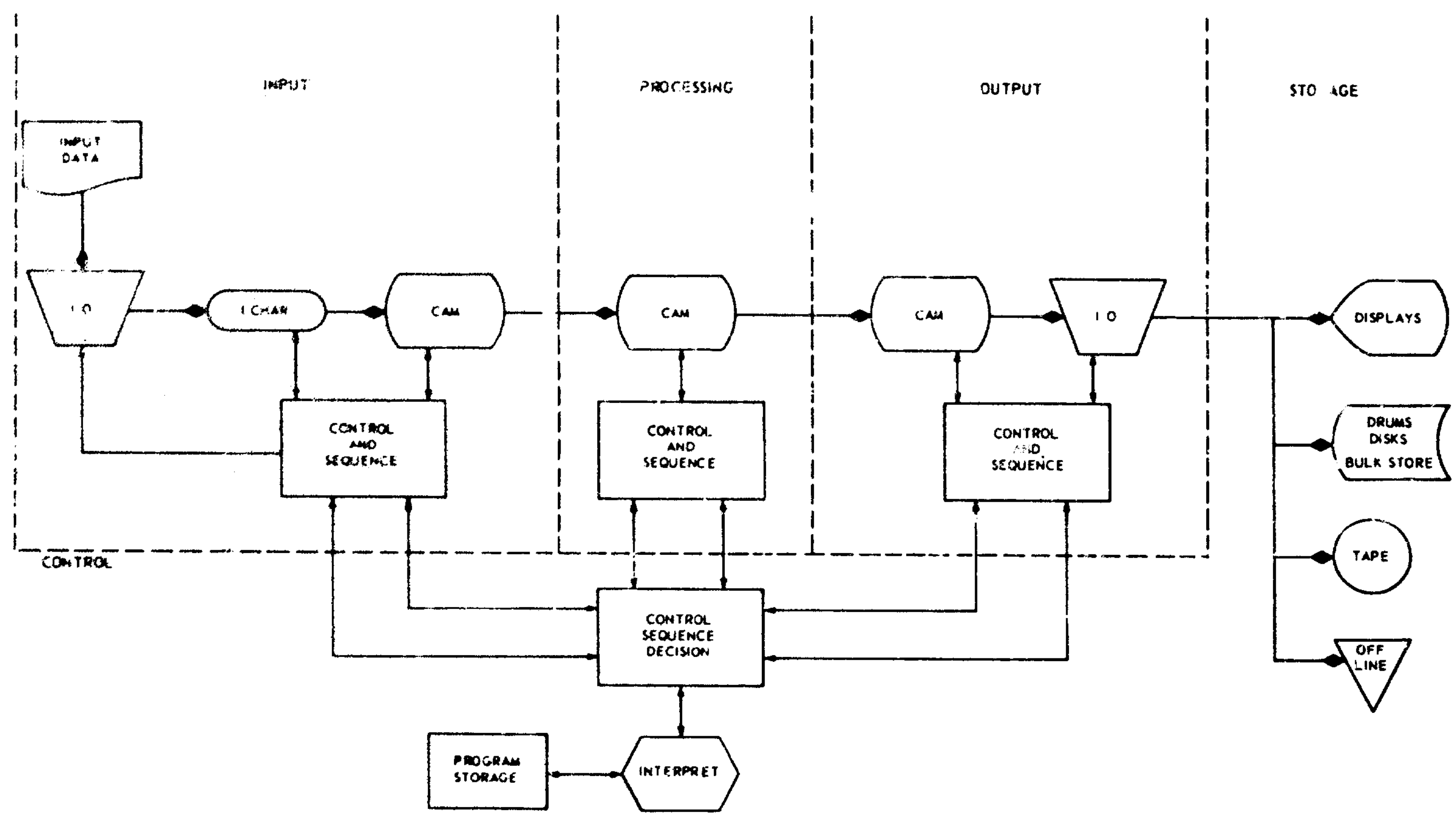


Figure 1. Data Extraction Processor Organization

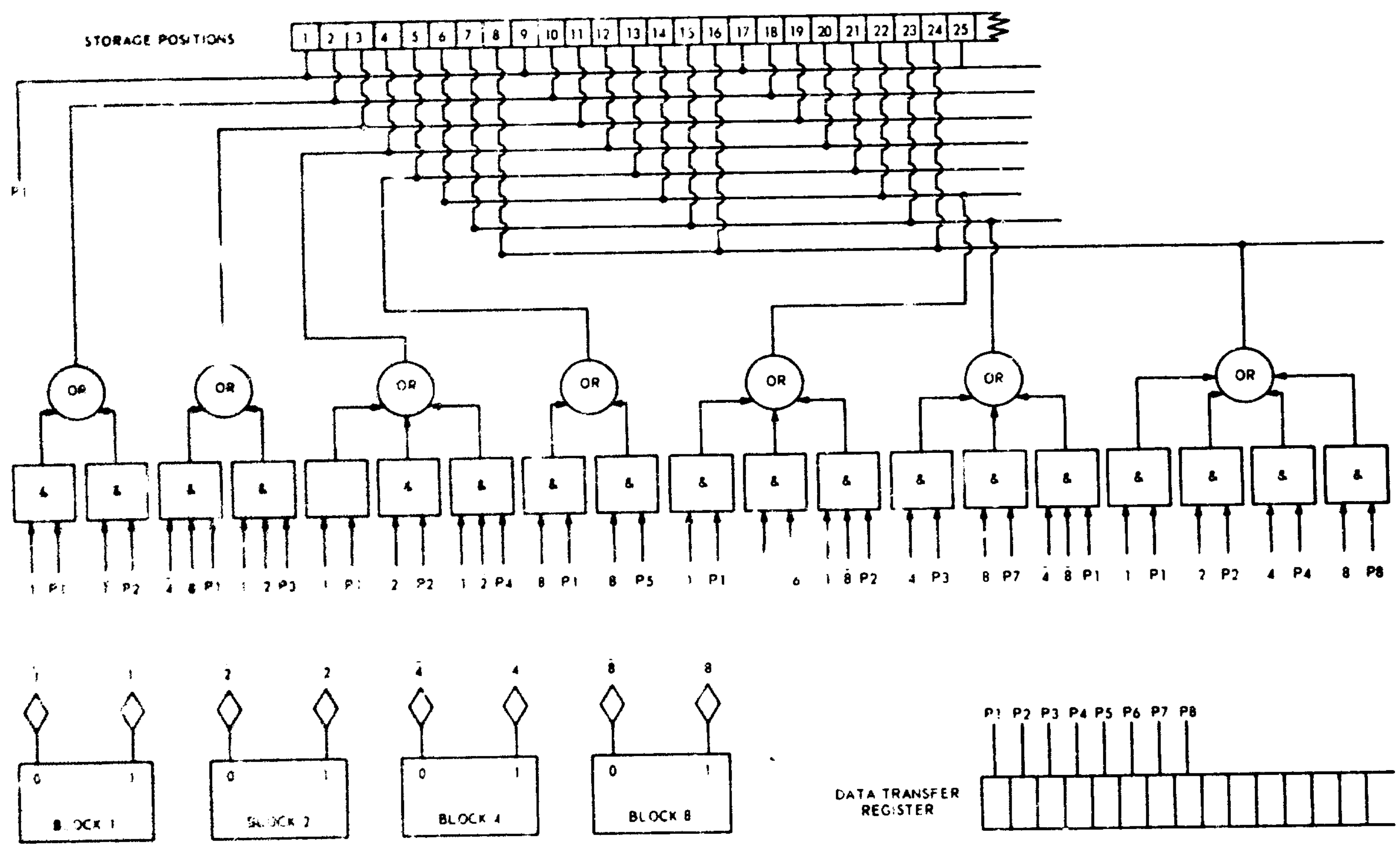
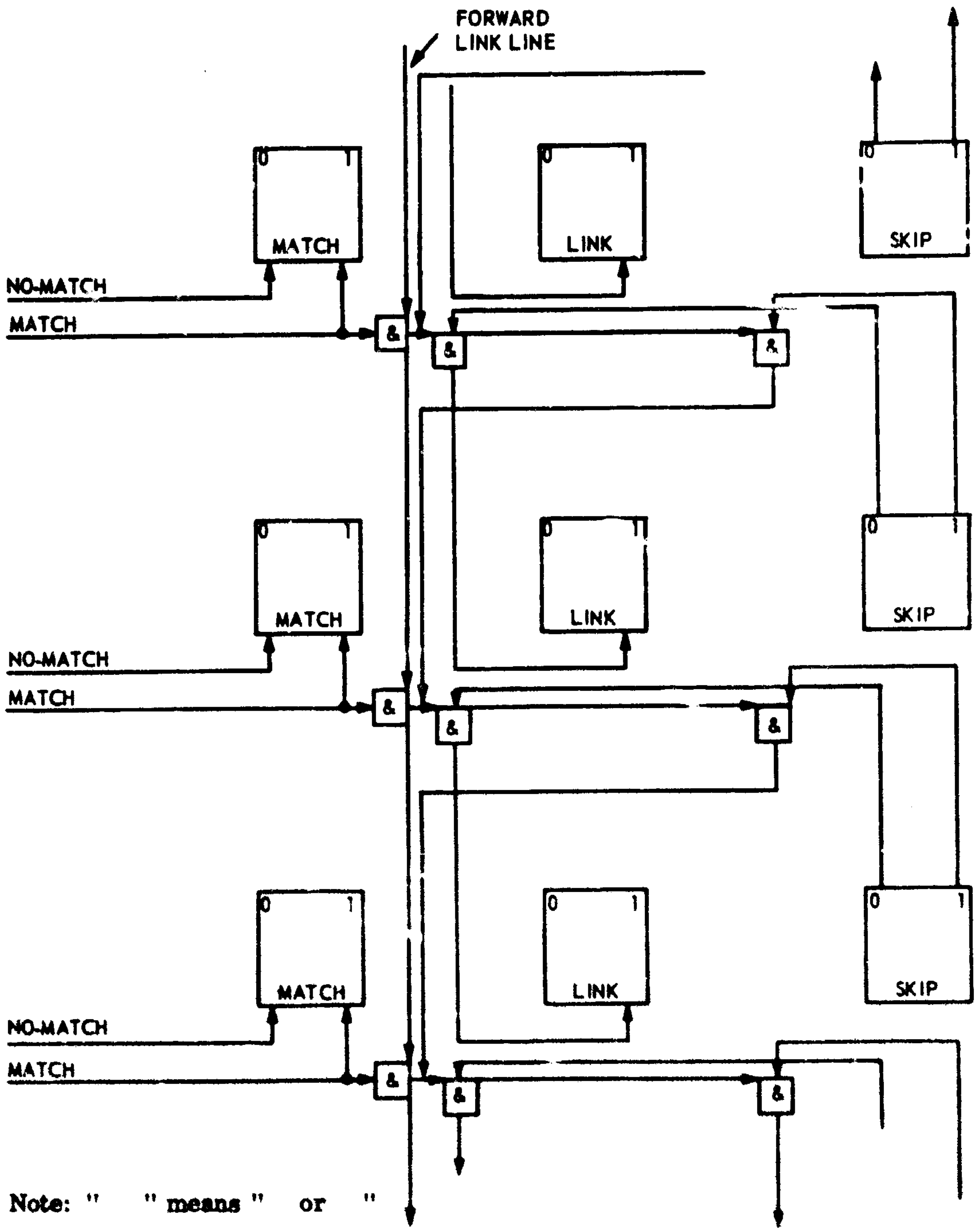


Figure 2. Block Circuitry for Connecting Storage Positions to Data Transfer Circuitry

programmer is not allowed to organize storage in more than one data length mode at a time. That is, in case the 2-character mode is selected, then all of storage is organized as two characters per combined register; and a new mode select instruction must be given if it is desired to transmit data other than two characters per memory reference. After the next mode select is given, the two-characters-per-transfer select is nullified. During the register mode operation, the actual grouping of the characters is determined by machine circuitry which forces the groups of characters to start in fixed positions relative to the first position of the core memory.

In addition to the marker bits of the memory, each storage position contains logic for use in connecting adjacent positions for logical manipulation by the programmer. This circuitry serves as an instantaneous chaining device to indicate the immediate preceding or following storage position.

Each position contains a MATCH indicator, a LINK bit, a SKIP bit, and a START-OF-CHAIN bit. When a COMPARE is found in any storage position, the MATCH is set in that position, and a signal (link carry pulse) is sent to the chaining bit in the adjacent position (provided the storage has been set for the link operation). If linking is "left" or "before," the link carry pulse goes immediately to the preceding storage position; if linking is "right" or "after," the link carry pulse goes to the subsequent position. A link carry signal arriving at any storage position is gated with the skip bit. If the skip bit is set, the link carry signal is forwarded to the next storage position in the direction of linking (see Figure 3).



Note: " " means " or "

Figure 3. Link Between Registers (Only Forward Direction Shown)

During the match and link procedure, special track is kept of the next potential character in the sequence. This is necessary to ensure the capability of nullifying all entries of the chain in which a failure to match occurs, and so eliminating a false chain indication. Thus, when at the end of a compare cycle a 'next successor' character exists which did not match the expected character, a pulse is generated to nullify the link to that character and to all other characters in that chain up to and including the first character in the chain. The first character in the chain is tagged by a START OF CHAIN bit while the next chain candidate is marked by a FOLLOWER bit. Figure 4 illustrates the sequence that the cancellation pulse executes in nullifying the chain.

Let the START-OF-CHAIN bit be denoted by S, the FOLLOWER bit be denoted by F and the MATCH bit by M. Then, in a search for the sequence ABC, consider what happens to the string ABCAD in memory. The linking control circuitry is not shown in Figure 4 and is assumed to be set up for the link right function. Similarly, the carry propagation is not shown for the normal carries from cell to cell. The first portion of the match cycle marks the 'A's with the M bit and the S bit set to 1. A strobe pulse on the S-set line allows the S bit to be set with the M bit. The F bit is then set, through the delay, to one in the following memory cell. (Subsequent matches for other characters will not affect the S bit, since the search for 'A' initiates the chain.) The second match cycle looks for a cell with a 'B' and an F bit set to 1. For those cells, the match bit is set and the F bit reset to 0. The setting of the F bit in the next register is accomplished through a delay which enables the match or no-match line to come up in the next cell, and guarantees the correct timing

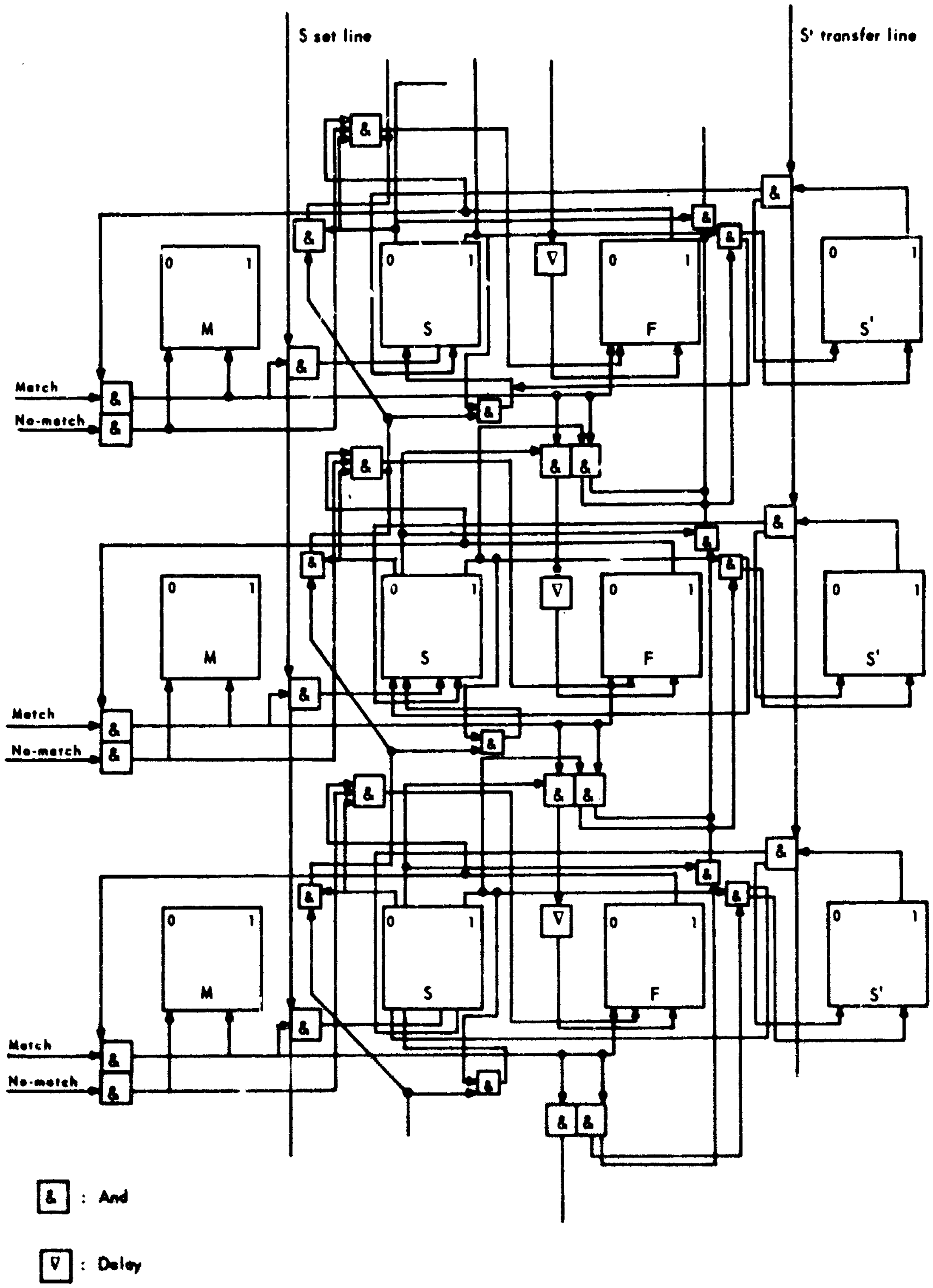


Figure 4. Link and Erase-Mismatch Logic

in the examination of the match and follower bits. The S bit is also examined. If the S bit is set to 1, a special carry is propagated upwards in the chain to set the preceeding S bit to 0 and the associated S' bit to 1. If the S bit is 0, the normal setting of the F bit occurs.

For those registers in which the no-match line is up and the F bit is set to 1, a no-match procedure is initiated. If the S bit is 0, a no-match pulse is propagated to the preceding register. The same examination of bits occurs in each succeeding cell until a cell for which the S bit is 1. The propagation ceases at this point. The S bit for that cell is reset to 0.

At the end of the comparisons, only those strings in memory which exactly matched the desired one are marked. The beginning of each is marked with the S bit set to 1, the end character is marked with the M bit set to 1, and the end character plus one is marked with the F bit set to 1. All non-matching strings have had their S and F bits reset.

In case the partial matches overlapped at some point, indicated by the attempt to set the F to 1 in a cell for which the S was already 1, the initial points of such strings were 'saved' in the S'. Provision is made to transfer the contents of the S'plane to the S plane and to initiate the comparisons of these strings once again. At the end of the examination for the string, ABC, the memory will appear as:

Character	S-bit/F-bit/S'-bit/M-bit			
A	1	0	0	0
B	0	0	0	0
C	0	0	0	1
A	0	1	0	0
B	0	0	0	0
D	0	0	0	0

The input editing processor is conceived as 256 words of memory and is used to do any upper-case or lower-case conversions on the input. This might consist of editing teletype input and converting it from the 5-bit code to the 8-bit internal code of the computer. All upper-case characters such as numerals, capitals and punctuation would be handled at this time. Certain format controls would be recognized in this section and appropriate controls generated. Paragraph and section indicators fall in this set.

The general search processor will contain on the order of 64 thousand words of storage (one character per word) and will execute the programs for the data extraction process. These include identification, extraction, routing, indexing and abstracting. All data manipulation for the data extraction process is done in this section.

The output formatting processor has an eight-thousand single character memory and is used for formatting and outputting the data passed to it from the general search processor. Its function is to make up entries for formatted files, for reports of various kinds, and for ordinary output of the nature in current usage. This processor makes up the controls for the display devices which may be attached to the processor as well as the output units such as disks, drums and tapes. Figure 1 illustrates the overall organization of the computer and the relationship of the segments to the design.

Each subunit of the processor has its own mask and compare register pair. These operate in a somewhat conventional manner. That is, for each bit set to a 1 in the mask register, the corresponding bit in the compare register is matched against the core storage in the manner specified in the instruction.

For the input editing processor subunit, the registers of this pair are of the same length as that of the storage positions. The general search subunit, however, has a pair considerably longer than the individual storage position. This is to ease the problem of lengthy compares of adjacent strings of characters. Thus, if a string of N characters is to be compared to storage, then as much of it as possible is loaded into the mask and compare pair before the match operation commences. The processor is able to automatically sequence through strings of this nature in the comparison matching, first against the first character and then shifting and matching against the remaining characters one at a time until no match is made or until the number of characters specified is completed. This type of operation is used primarily in conjunction with the match and link operation previously discussed.

Another property of the linking feature of the associative processor is the skipping of characters determined to be nonrelevant to the current search. This is accomplished by means of the skip bit associated with each register in memory. When the register is flagged as being nonrelevant (to be skipped) and a match and link operation has identified the adjacent register as being matched, the link carry pulse is propagated through the position marked by the skip bit. It is gated by the skip bit to the next register and so on until a register is encountered for which the skip bit is not set. This register is then 'linked' to the original register. As this process may occupy some time, a delay is made until the propagating carry pulses are all ended. By this means, occurrences of common sequences of line feeds, carriage returns, etc., of teletype can be ignored where desired, in the search for significant data. In the situation where blanks between

words (text words) are not significant, these may also be skipped and the problem of accounting for one, two or more blanks legitimately appearing between words is avoided at a saving in processing time.

The overall supervision of the various sections of the processor is by the central supervisor. This supervisor acts in a like manner to that of a multiprogramming computer. The supervisor controls the data flow between modules of the computer and coordinates the activities of the components. Communications between portions of the system pass through the supervisor module and then to the recipient component. The individual components of the processor contain their own central processors which interpret and execute the instructions for that module. The I/O commands are retained in the supervisor for interpretation and execution while the internal memory commands are executed directly by the modules.

Associated with the central control processor is a program storage memory in which the program elements of the operating programs are stored. As each module becomes free to execute an instruction, the central computer fetches the next instruction for that module. Program execution for the various modules of the computer is interleaved logically and overlaid in execution time. In this manner, while one section of data is being set up for outputting, the next section can be formed for the next extraction process.

Each memory processor unit has a mask-compare register pair for the associative search operations. The comparison field length is the same as that of the storage register length and overlies the marker bits. This allows for associative compares on the results of previous scans. Bits to be set during the

scan are specified by the instruction. The mask register enables the user to compare for exact match on any subfield of the compare register desired.

The compare register and its associated mask register hav additional positions available for the storage of characters. Associated with these extra positions in a count field that may be loaded with a count of the number of characters currently in the extra positions. This count and the added register length are used in the iterative comparison instruction for the main processor in which the data extraction searches are made. The input editing processor module and the output formatting processor module are restricted with respect to this special feature.

In addition to the ordinary functions of the digital computer that are used in control for branching, indexing and so on, certain other functions are uniquely restricted to the associative memory processor. The normal, so-called associative instructions apply to all of the elements of the processor. These include such things as EXACT MATCH, READ FIRST MATCH, WRITE IN MATCHED LOCATION. Each of the processor modules has, in addition to these instructions, a special match and link operation, link compare, which allo s registers other than those for which the match condition is satisfied to be marked for retrieval of other manipulation. The main data extraction module has a modified exact match instruction suited to the problem of searching for lengthy strings of data in an automatic fashion. This instruction is called CYCLED COMPARE. These are implemented either by hardware or by micro-programs stored in conjunction with the central control processor and transmitted to the individual associative processor involved.

Cycled Compare

The contents of the comparison register are compared to associative storage. The first position in the register is matched against core memory and the start-of-field bit is set in all matching memory registers. Simultaneously, the mask register bits inhibit comparison in bit positions containing a in the mask register.

For all registers in memory which matched the compare register, a link carry pulse is generated and forwarded to the next character in memory. The mask and comparison registers are shifted left one character position, and the processor executes the next cycle in the comparison. This character is matched against the memory in the same manner as the previous character, but with the additional requirement that the matched character in the memory have the link bit set to one. The process of shifting and matching continues until a no-match pulse is received from the memory or the character count for the compare register is 0. Automatic delays are made at each point that a no-compare signal is required to eliminate a partial chain in memory.

Link Compare

The contents of the compare registers corresponding to 1 bits in the mask register are compared to each register of the memory. The match bit is set to 1 in each register in which the contents match exactly. Registers next higher or lower in the memory, as specified by the program, have the link bit set to 1. The skip bit in these registers is examined and the link carry is propagated to the next register whenever the skip bit equals 1. The link bit is not set to 1 in

those registers for which the skip bit is set. For example, assuming the match condition is an 'A,' then the contents of memory might be affected as shown below for the link right operation.

Prior to LINK COMPARE:

Register contents	M-bit	L-bit	Skip-bit
X	0	0	0
A	0	0	0
B	0	0	1
R	0	0	0
A	0	0	0
Y	0	0	0

Subsequent to LINK COMPARE:

Register contents	M-bit	L-bit	Skip-bit
X	0	0	0
A	1	0	0
B	0	0	1
R	0	1	0
A	1	0	0
Y	0	1	0

EXAMPLE PROBLEM

Because of the complex nature of the problem in data extraction area, the sample text is defined descriptively rather than in a tabular array of statistics. The type of document to be processed is the daily summary report of the type that is sent from regional office of the FBI to the central headquarters in Washington, D. C. This report details the activities of criminals in the western part of the United States for the previous day and includes such information as the type of activity, the number of persons involved, the location of the activity, and any results of police efforts.

The data is to be input to the processor and significant data relative to the preceding items is to be extracted for insertion in a formatted file. The report[1] has several sections: header, classification, criminal activities, and end of message code. The portion to be examined is the criminal activities section and all of the information in this section is to be considered significant. The criminal activities section is made up of individual reports of activities.

The report begins with the header information which identifies the originating source and contains the agent's name submitting the report, information converning the recipient, and the location of the recipient. The heater is terminated with the synbols 'BT' preceded and followed by the codes < ≡ ≡.[2]

The criminal activities section of the report is initiated with the line CRIME REPORT (CRIMINAL ACTIVITIES). This is followed by <<≡. The second line on this section contains the report number, and the number of the day of the year in which the report is generated. The remainder of this section is separated into paragraphs,[3] with each paragraph containing a number of sentences describing individual activities.

---

[1] Figure 5 is an example of this type of report. The first portion shows the printed output while the second shows the characters necessary to control the printer. These characters are the input to the data extraction processor and are the ones from which the data is to be extracted.

[2] These codes are shown in the second portion of Figure 5 and are carriage return, line feed, line feed, respectively.

[3] In the sample shows as Figure 5 the paragraphs are labeled '1. MIDWEST and '2. WEST.'

FBI<<≡DE BOND 007 21/10162<<≡OX 2423172 ZEX<<≡FM FIELD OFFICE
5<<≡TO FBIWASH/RFK<<≡CHICAGO<<≡≡BT<<≡≡C O N F I D E N T I A L<<≡
≡≡SECTION 2 OF 2<<≡≡CRIME REPORT (CRIMINAL ACTIVITIES)<<≡TO FBIW
ASH REPORT 031<<≡≡1 MIDWEST: ILLINOIS AND INDIANA:<<≡≡ A. 18
35H 14 JAN. CHICAGO. CR ENTERED CONSTRUCTION SHACK<<≡VIC IL 087
415, STOLE DYNAMITE. (THEFT)<<≡≡ B.2016H 16 JAN. GARY. CR ROBB
ED MOTEL RESTAURANT VIC IN<<≡385 012. STOLE CAR AND HEADED EAST.
LOSSES: $900, 1 VHE.<<≡(ARMED ROBBERY)<<≡≡ C. 2100H 17 JAN. C
HICAGO. CR BOMBED NEGRO CHURCH VIC IL 440<<≡087. LOSSES: 3 KIL,
2INJ, 2 VEH DAM. (BOMB)<<≡≡ D. 0830 17 JAN. FORT WAYNE. 3 CR R
OBBED BRINK ARMORED CAR<<≡VIC IN 042 767. POLICE APPREHENDED AT
ROAD BLOCK. LOSSES:<<≡$2 MILLION, 2KIL(1 CIV, 1 POLICE). CR LOSS
ES: 1 KIL, 1 INJ,<≡ THIRD MAN ESCAPED, MONEY NOT FOUND. (ARMED R
OBBERY)<<≡≡<< R. 2030H 20 JAN. INDIANAPOLIS. 2 CR ATKD UNESC W
OMAN <<≡VIC IN387 088. (SEXUAL ASSAULT)<<≡≡ F. 1917H 18 JAN.
JOLIET. EST 20 CR ATKD AND BURNED POLICE<<≡CAR VIC IL 375 221.
OFFICER INJURED, NO ARRESTS. (ASSAULT)<<≡≡2. WEST: SOUTHERN CALI
FORNIA:<<≡<<≡ A. 1600H 17 JAN. LOSS ANGELES.2 CR GANGS RIOTED
IN THEATRE≡<<VIC SC 122 048. (RIOT)<<≡≡ B.1425H 19 JAN. SAN D
IEGO. CR SNIPER FIRED ON CIVIL RIGHTS DEMONSTRATION VIC SC 312
064, KILLED MINISTER.<<≡(MURDER)<<≡≡ - - - - - - - -
- - - - - - -<<≡≡BT≡≡>NNNN - - - - - - - ≡≡≡≡

Figure 5. Sample Teletype Report as it Appears in Memory

SAMPLE TELETYPE REPORT

FBI
DE BOND 007 21/1016
OY 2423172 ZEX
FM FIELD OFFICE
TO FBIWASH/RFK
CHICAGO

BT

C O N F I D E N T I A L

SECTION 2 OF 2

CRIME REPORT (CRIMINAL ACTIVITIES)
TO FBIWASH REPORT 031

1. MIDWEST:ILLINOIS AND INDIANA:

A. 1835H 14 JAN.CHICAGO. CR ENTERED CONSTRUCTION SHACK VIC IL 087 415, STOLE DYNAMITE.(THEFT)

B. 2016H 16 JAN. GARY. CR ROBBED MOTEL RESTAURANT VIC IN 385 012. STOLE CAR AND HEADED EAST. LOSSES: $900, 1 VEH. (ARMED ROBBERY)

C. 2100H 17 JAN. CHICAGO. CR BOMBED NEGRO CHURCH VIC IL 440 087. LOSSES: 3 KIL, 2 INJ, 2 VEH DAM. (BOMB)

D. 0830 17 JAN. FORT WAYNE. 3 CR ROBBED BRINK ARMORED CAR VIC IN 042 767, POLICE APPREHENDED AT ROAD BLOCK. LOSSES: $2 MILLION, 2 KIL(1CIV, 1 POLICE). CR LOSSES: 1 KIL, 1 INJ. THIRD MAN ESCAPED, MONEY NOT FOUND. (ARMED ROBBERY)

R. 2030H 20 JAN. INDIANAPOLIS. 2 CR AT KD UNESC WOMAN VIC IN387 088. (SEXUAL ASSAULT)

Figure 5. Sample Teletype Report as Printed

F. 1917H 18 JAN. JOLIET. EST 20 CR ATKD AND BURNED POLICE CAR VIC IL 375 221. OFFICER INJURED, NO ARRESTS. (ASSAULT)

2. WEST: SOUTHERN CALIFORNIA:

A. 1600H 17 JAN. LOS ANGELES. 2 CR GANGS RIOTED IN THEATRE VIC SC 122 048. (RIOT)

B. 1425H 19 JAN. SAN DIEGO. CR SNIPER FIRED ON CIVIL RIGHTS DEMONSTRATION VIC SC 312 064, KILLED MINISTER. (MURDER)

BT

NNNN

Figure 5. Sample Teletype Report as Printed (Cont'd)

The following discusses the procedure used in identifying those portions of a document that are of significance to the user. The analysis will proceed part by part with a description of the parameters necessary to describe the data and the permissible variations. For ease of examination, the second portion of Figure 5 has the character blank (ƀ) shown as a space between characters rather than printed as 'ƀ'. In the following discussion, the symbol '1' should be taken to mean 'any letter' and the symbol 'd' should be taken as 'any digit.' This notation is used to describe the configuration of data fields whose exact contents are not known but whose characters may be characterized in some predetermined manner.[1]

<u>Criminal Activities Section Identification.</u> The criminal activities section is headed by a line whose first two words are crime report. This appears to the processor as <≡CRIME REPORT. The characters, <≡, indicate that those characters which follow begin a new line in the printed output. This sequence is the standard for the beginning (or end) of a line in teletype practice. It is subject to some variations due to the desire for control of line spacing, operator habits, etc. Thus, <<≡≡, <≡≡≡, <<<≡ are all equivalent symbols for this configuration. The presence of a single < or ≡ is not enough to justify the conclusion that the line marking configuration has been found since it is a simple 1-bit transformation to get from some other legal text character to either the < or the ≡.

---

1 The reader is referred to "Anexor", C.H. Forbes, 31 Jan. 1963 for more detailed description of search parameters.

Thus, the report is scanned from the beginning until the configuration < ≡ ≡ or <<≡ or <<≡ ≡ is found. Other equivalent configurations of the line marking characters are accepted. Examination is made of the characters immediately following the line marking characters and identification of CRIME REPORT, completes the search for the start of the criminal activities section.

Report Number Identification. This field consists of the word REPORT followed by any 3-digit number in the numeric range 001-366. This field will be followed by the line marking string either directly or after one or more intervening blank characters.

Paragraph Number Identification. This field is always numeric and always appears at the start of a new line of text. Since there are other numbers that could begin a line in the report, it is necessary to define more closely the exact configuration of this information. In this example the highest number that appears is 2; however it is possible for the number of paragraphs to exceed 10. It is unlikely that the number of paragraphs will exceed 99, so that there is an arbitrary restriction that the number be either a single digit of two digits. The paragraph heading line also contains colons (:) to separate fields and is the first line to contain these characters. Therefore, the first line containing the requisite numeric information and the colons will be the paragraph heading line.

Date-Time Group Identification. This group is the first in the sentence and is followed by a period . . The field is usually three strings in length, however, this is not a requirement since the time portion of the field is sometimes absent. Since the output of the search requires that the information be separated

into time information, date information, and month information, the separate fields of this group will have to be identified.

The entire date-time group is confined to the area beginning with the start of a sentence and running to the first period. The month is stated as three contiguous alphabetic characters (it is possible to list all of the month configurations, but restricting the information to the name of the month obviates the need to list that much detail). The day of the month is any one- or two-character field consisting of digits bounded by blanks and immediately preceding the month field. These fields are required to be present in the report, but the time field may or may not appear. If it is present, it will appear before the day of the month subfield and will consist of four digits by 'H.' The entire set is bounded by blanks. Other fields and subfields that appear in the report are

1. Location
2. Activity Category
3. Subject (usually CR for criminal etc.)
4. Activity (verb ending in ED or the word ATKD, etc.)
5. Map Coordinates
6. Object (object of the activity)
7. Losses Information (descriptive material listing damages incurred)
8. Losses Information Embedded in Miscellaneous Statements (this material is narrative in nature and such words as KILLED or INJURED or CAPTURED appear as clues to the location of such data)
9. End of Message (the character sequence < ≡ ≡ NNNN)

DISCUSSION

Due to the complexity of the type of data to be manipulated, it is impractical to make an explicit comparison of the processing times for an associative processor and a conventional processor. Indeed it is difficult to specify just what the actual processing time will be on any given processor for this problem. Therefore, the evaluation of this design will be a general discussion of the difficult areas of processing and the manner in which the design approaches a solution of the problem area. One of the major areas of the problem is examining each character in the entire message string to find all occurrences of strings of characters that denote the boundaries of words. Among the characters which denote the boundary of a text word are blank, comma, colon, semicolon, etc. Combinations of characters, such as period, blank, also indiate the word boundary condition as being present. To test for a word boundary, it is necessary to query each incoming character to determine if it is one of this set. The conventional processor may make this determination more or less efficiently during the input phase, but the associative processor needs only one comparison. In searching for all occurrences of some string of characters, the program in the conventional processor must compare each possible character as a potential starting point for the desired character string. The associative processor may mark all such strings regardless of their starting position in one set of comparisons for each of the characters in the sought-for string. This might mean that in the case of a message of 1000 characters, the conventional processor would require 1000 comparisons while the associative processor will need but one. Of course, some pre-editing can be made to parameters of the search so as to construct a thesaurus of the text during the

input phase; but if the parameters are extensive, this time may be prohibitive. In the associative processor, even such pre-editing is alleviated since the search within the parameters for the simplifying data may also be made associatively. Thus, some of the savings possible by pre-editing the parameters for a conventional processor are also available to the associative processor. These may be incorporated into the pre-editing processor where the editing for word boundaries is done in parallel with the search for the significant data.

The overall gain possible for this type of organization over that of the conventional processor can range up to several thousand to one in special cases where many items of similar structure are in the memory at the same time, and where many searches may be made for each loading of the memory with data. The lower limit comes when the parameters are either very simple in nature and the conventional processor can accomplish the task on the fly, or the data is so variable that by the end of one or two searches the data being examined is reduced to only one possible match. In this case, further searches, although capable of searching the entire memory, are actually restricted to only one register. For this type of processing, there appears to be a time advantage of one order of magnitude.

Section III

DICTIONARY LOOK-UP

PROBLEM STATEMENT

The dictionary look-up problem covers the identification of words in tex solely by means of a dictionary in order to prepare for further processing o the text. The text involved may originate from a variety of sources ranging from such strictly formatted material as file entries from a formatted file t text published in newspapers, books, and periodicals. The goal then, of the dictionary look-up process is the identification and marking of the text worc in a manner to facilitate future processing. The marking may include such things as parts of speech, index term classification or other, special prope: s of value to the system user. Words not found in the dictionary should be fl: d so that separate processing of these terms may be simplified.

To allow practical study, a limitation to this procedure is the restrict; that all information to be derived and appended to the word is to be derived m an examination of the word and its characters separate from its context. utilization of information about the neighbors of the word in question is t to be considered. Only that information stored in conjunction with the di ry entry is allowable for classifying the word as to its use in later processin

The data input for the dictionary look-up processor problem will gene be considered to be textual in nature. The exception as previously noted formatted report. It may be derived from teletype input or from typesett out

from the publishing industry. No special characters other than those normally used in text will appear to designate the break points between words or phrases. The assumption is that the word boundaries will be formed primarily from such items as the characters "blank", "comma" "blank", "period" "blank", "colon" and "semi-colon". Other characters and configurations of characters appear in the set and depend upon their usage in the specific text sample for membership in the set.

CHARACTERISTICS OF THE PROBLEM

The processor will be required to accept textual data in the form of strings of characters. There will be no special characters present other than those normally in text to denote the separations in text of paragraphs, lines, sentences or even of words. The location of the boundaries is of special interest in this case since it is just this problem upon which the efficacy of the system depends. As previously stated, the recognition of these boundaries will be one of recognizing character configurations that frequently are used for the purpose of defining the point of separation of one word from another.

The processor must be capable of looking up an incoming word or textual unit defined by the user and affixing to it such information stored in the dictionary previously. The look-up procedure must be capable of handling the variable length text units that may be of interest without being dependent upon the register length of the processor. This is necessary since for any length of register, measured in characters, there will be requirements that exceed that length. Although words in common usage average about eight characters in length, there

are single words on the order of eighteen to twenty characters. There are also compound terms which exceed this limit used in specialized areas such as chemistry, botany and medicine. This range of data must be handled in the processor.

The incoming string of characters to the processor will be accepted one at a time, and it will be assumed that the time for a comparison in both the conventionally organized processor and the associative processor will be equal to the time to transmit one register full of characters. Since it is possible for input data to be saved until a previous job is completed and then transmitted to the processor at a higher rate, it will usually be possible to have the input data rate equal to the compare or any other basic machine cycle desired.

The capacity of the main storage unit will be assumed to be sufficient to contain all of the terms and their associated data from the dictionary. If the dictionary exceeded the size of the main memory, the performance of the processor would be degraded by the data transmissions to and from the memory unit. This will be a restriction on the problem and will be considered in conjunction with the problem of text statistics. (See Section IV.) It is appropriate to note at this point, however, that approximately 97% of words occurring in a large sample of news material were accounted for by 44,821 dictionary words (see Table I). It is further assumed that the data looked up will be outputted to another processor for further processing or that there will be sufficient storage to contain a moderate amount of the current data after processing. This would be enough for a single intelligence message or news story or a small document.

The method of handling the data is as follows. The text characters are read into the memory and the boundaries of the words are located. The words are searched for in the dictionary and those words for which a match is found are tagged with their information as stored in the dictionary. The remaining words are tagged as not being in the dictionary and then the entire set is outputted for further processing or stored in the appropriate section of the memory to be utilized at a later time.

PROCESSOR DESIGN

The dictionary processor design is based on the assumption that the data flow into the system is restricted to a single input at a time. This requirement is reasonable since it may be assumed that the input may be buffered to achieve the maximum transfer rate that the central processor can accept. Therefore, if the data transfer time is equal to the minimum time to form a comparison with the incoming data, no further gain is possible by increasing the data transfer rate. Any scheme for increasing the throughput speed of the processor must, therefore, address itself to the problem of increasing the effective rate at which the input can be handled. This could be accomplished by implementation of even more parallelism.

The design is separated into a system control unit, an I/O system, and a memory system (Figure 6). The system control exercises control over the data flow internal to the system as well as all I/O flow. The I/O system consists of the I/O units and the associated controllers, channel devices, etc. The memory system consists of two associative memory units and their controlling

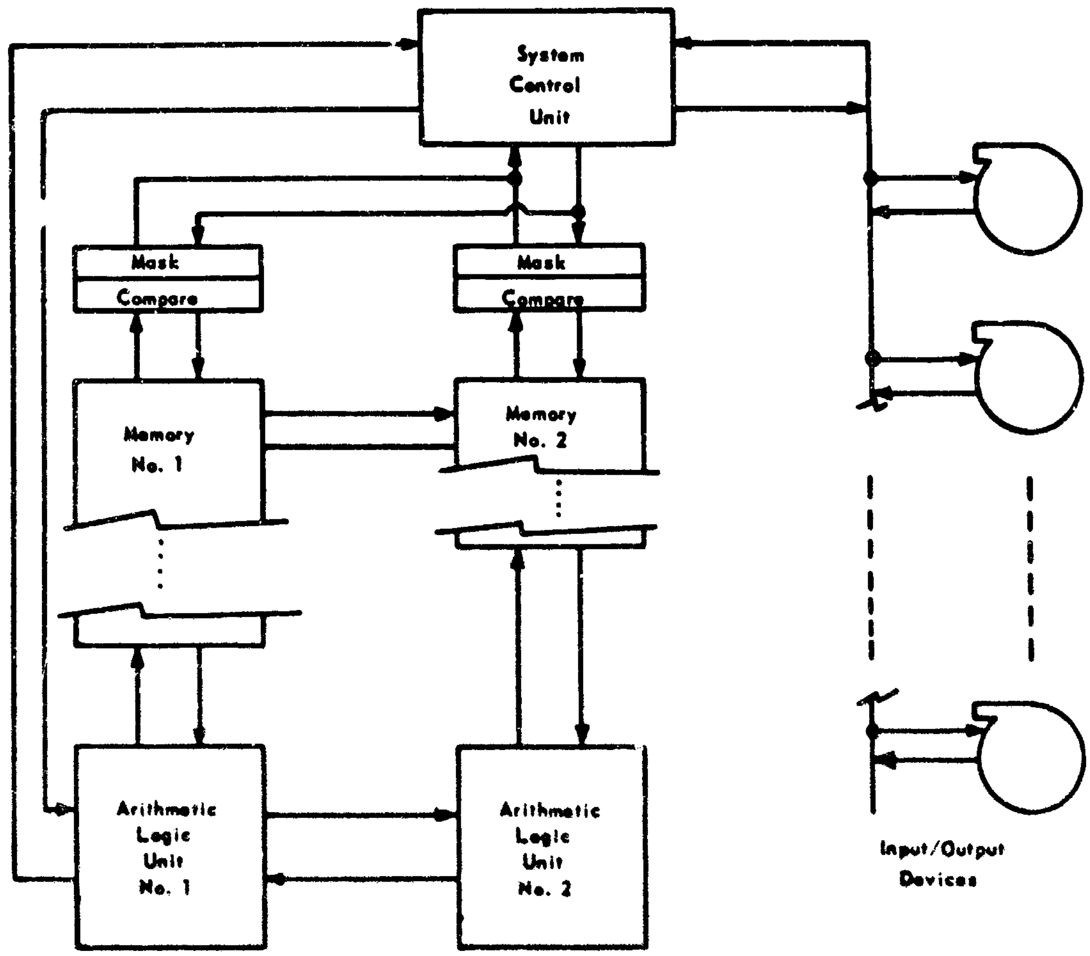


Figure 6. Machine Organization for Dictionary Processing

arithmetic-logic units. All data transfers are under control of the system control unit. Program flow is monitored from the system control unit and initiation of branching to new or different segments of the process is supervised from it. The individual instruction sequences are decoded, interpreted and executed in processor memory modules. These are the main processor memory and memory control unit and the channel processor memory and memory control unit. The data that is to be processed is retained in the memory units of these modules until it is needed.

The I/O devices are controlled by the system control unit and requests for data from either of the two memory-processor units are relayed to the I/O devices.

The memory register length in both of the memory units is sufficient for seven characters plus some control bits. This length is a compromise between a long register in which any of the possible strings of data might be stored completely and a short register which would allow for the minimum amount of storage to be used for a given set of terms. The control information relates to the data that is required as a result of the look-up process. This might include such things as the class of word found, the possible affixes available, status as an index term; etc.

The smaller of the two memory processors, to be referred to as the channel processor, has storage capacity on the order of 500 registers. This allows for the storage of all the legal characters for any run as well as a set of terms determined to be the highest frequency set with regard to the text being processed. These terms, in the context of natural English text, are common to a

large degree to many samples of text that have been studied. Some of these words are 'to', 'of', 'by', 'for', 'a', 'an', 'the' and 'but'. The list will contain approximately 100 terms, most of which will be common to any sample of text. There will also be capacity in the memory unit for extra terms not included in the aforementioned set that are specially adapted to the particular data sample under consideration.

The individual characters are used in a word boundary algorithm so that the boundaries of the incoming words are recognized.

The channel memory processor is envisioned to be at least one order of magnitude faster in its execution of instructions than the main processor. This will allow for the inspection of each character in the data stream as it is inputted as a word to the channel processor. Words will be formed up in a buffer register before transmission through the central processor unit to the main memory processor unit. For words whose length is less than or at most equal to that of the longest of the high frequency words, a special look-up is taken prior to passing the word into the main memory. Any of the words that match are flagged before transmission to inhibit the look-up procedure in the main processor. Since the channel unit is one order of magnitude faster than the main unit, the look-up just described will be made at no cost to the overall data flow time. Indeed, there will be a net saving in time in the process since the look-up of these words in the main memory will be eliminated. The actual amount of savings gained will depend upon the characteristics of the text material. In one case, the number of words accounted for in this way was approximately 45%. Thus, a higher speed I/O unit may be used, taking advantage of the speed gain possible with this pre-editing look-up procedure.

The two associative memory units of this system are of radically different sizes; however, the functions are quite similar and overlap to a large extent. The larger of the memories has registers to contain approximately 50,000 dictionary words and their relevant data. The mask-compare registers of this memory are of the same length as the memory registers and allow for the inhibiting of compares on any specified subset of the word.

The operations of the two associative memories are quite basic in nature and consist of the usual compare for exact match (with the match register for bit selection), read out first match, write in first location, multi-write and link between registers. The link between registers is limited to the following register only (although the same procedure for preceding registers will be of value in other applications such as data extraction). The linking operates by setting a bit in the memory register following the register in which the exact match condition occurs. That is, for any register in the memory that is matched with the contents of the comparison register, the next adjacent register in the memory unit will now be marked with an associatively addressable bit. Read or write operations will then be performed on the basis of continuing matches so that only the last register in the sequence of registers that has been matched will be marked whenever there is only one string in memory that matches the search string. This allows for the automatic resetting of the match bits in each cell at the beginning of each new compare cycle without sacrificing of the information gained in the previous cycle. Since the memory unit will have only one of each string and the data to be retrieved with the search is not the word but some information about the word, the cancellation of the partial matches

generated on the first set of characters results in a net gain in processing time. A multiple match condition, resulting from scanning for a partial string or from the storage of more than one of any item, will result in more than one match bit remaining set at the completion of the scan and will require program separation of the desirable results from the undesirable.

The register length for the processor memories is assumed to be seven characters and control information relating to the usage of the term and to its status in the system. This length was dictated by the storage requirements of the dictionary (average length of words, etc.) balanced against the increasing demands on the hardware of excessively lengthy registers in the associative memory.

EXAMPLE PROBLEM

In order to properly evaluate the design of a dictionary processor it is necessary to define dictionary and text typical of the general area under consideration. The parameters of the dictionary that need to be defined are: the number of words in the dictionary, the frequency distribution of the words in their alphabetic subsets, the frequency distribution of the words by their length and the distribution of the text words related to the dictionary entries; that is, how many and with what distribution of frequencies do words appear in the text sample to be studied.

A text sample and a dictionary derived from it were defined. The text was defined to have characteristics common to those of news publications. Since the problem is essentially serial in nature, only the characteristics of an incremental

sample were needed to evaluate a tentative machine organization. Thus, a block size of 8000 text words was used for the evaluation. This size is the size for a block of data coming in to be processed. The look-up times for each of the designs of the processors are to be calculated from the statistics of this parameter. As stated above, the assumption is made that the text input is approximately at the speed that the central processor operates. Thus the comparison between designs will be linear in nature and the sample size of 8000 words will be valid.

A set of data derived for a sample of publishing text was studied in order to produce a sample text description and dictionary description for the sample problem. The sample text was defined to be approximately 4.0 million words of unformatted text. The spelling of the terms was assumed to be correct and the total unique terms of the sample was 71,805. Of these, 39% or about 28,000 terms appeared as entries in a dictionary derived from Webster's Collegiate Dictionary.[1] This dictionary does not include technical terms or words that are or originated as proper names. It is limited to common nouns, verbs, adjectives, etc. The average length of the words in the 4.0 million word sample was 5.21 characters per word. The average length of the 71,805 unique words was 8.53 characters. The difference in lengths between the unique and total sample averages reflects the relatively high proportion of such short words as: 'to', 'by', 'for', 'but', 'and' and so forth. These words modify the distribution of

---

[1] Comdict: A Common English Word Dictionary. Private communication from M. Jones, June 1963.

the word lengths in a manner which skews it to the right and so reduces the average length.[1]

It is assumed that the dictionary for the look-up is completely contained in memory and that the percentage of the text words found in the dictionary is quite high. In the case studied, the percentage was 97.35 (see Table I). This represented a total of 44,821 unique words with a total occurrence in text of 3.9 million words. The words not in the dictionary amounted to 26,984 words with a total occurrence of .1 million words.

The text input for the processor is considered to be one character at a time. The size of the text units has been set at 8000 words per unit in order to evaluate the comparative times for the look-up procedure. The words and their boundaries must be identified during the input phase and the word matched against an existing dictionary stored in the memory unit of the main and satellite or channel processor.

Evaluation of the processors was based upon several organizations of the search procedure. The first procedure was the complete dictionary stored in the conventional processor as one entity. The incoming word was compared to the dictionary by means of a binary scan technique with the result that the average time per input word was 15.101 compares per word. Table II shows the timing breakdown for this process.

Since the high frequency dictionary accounts for such a large proportion (about 45%) of the total words, the dictionary for these words was separated

---

[1]See Table VIII, parts 1a and 1b of Section IV for similar information regarding the distribution of words in the same sample including error words.

from the main dictionary and words of the appropriate length scanned against the small dictionary prior to the scan of the main dictionary. Although this amounts to a penalty in look-up time for those shorter words not found in the high-frequency dictionary, this is more than offset by the gain made by not having to look up the high-frequency words in the main dictionary. These results are tabulated in Table III. The scan in both cases was binary in nature and resulted in a net increase in processing speed. The time per search was reduced to 11.879 compares per input word from text, again on the conventional processor.

The technique of hash addressing by randomizing the input word was used to generate an address for the dictionary look-up. This method results in the address of the first element of a chain of words in storage, each of which yielded the same random address. An examination of the chain would proceed in sequence until the word was found or until the last element of the chain was compared. An assumption of three compare times was made for the time of conversion to compute the address, while 1.9 words per chain for the main dictionary was used for the length of the average chain of dictionary terms.[1] The method was assumed to be especially effective in the look-up procedure for the high frequency dictionary terms when they were stored separately. In this case, the chain length was assumed to be 1.0 words since the list is quite short.[2] When either of these values is exceeded, the time to process rises and the overall process time increases.

---

[1]Dictionary Techniques by J. B. Ratchford. AIDS/SAC Working Paper, 31 May 1962.

[2]See Table VII under Special Dictionary.

This method was applied to the scan for the high frequency dictionary and for the combined dictionaries, again on a conventional processor. The results are tabulated in Tables IV and V, respectively. The outcome of orga ng the data together for a combined scan by the hash addressing rather than h addressing for the high frequency dictionary and binary scan for the main tionary as in Table IV is a reduction in number of compares per input wor from 10.333 per word to 4.570 per word.

As a bench mark for comparison of the efficacy of the organization of associative memory into the two parts rather than considering it as one i computation was made of the number of compares per input word for the dictionary contained in the main memory. This is recorded as Table VI. e overall time for comparison averaged out to be 1.625 comparisons per in word or 36% of the best possible time for the conventional processor. W he channel memory processor is added to the associative processor, the nu of comparisons per input word drops still further to .958 compares per i word, which is 21% of that for the conventional machine.

DISCUSSION

The conclusion to be drawn from this processor study is that an ove gain of about 5 to 1 is possible when the associative processor is compa the conventional processor. This margin will increase when the proced or the search in the conventional processor varies from the optimum given e hash addressing parameter estimates. The main gain to be obtained in processing lies in the speeding up of the effective input rate of the text ans

of pre-editing on the fly in a channel processor. The pre-editing may be obtained at little or no cost in time by utilizing few, high speed components and a small, very fast memory unit.

Since the procedure is essentially linear in nature, the separation of the word identification and initial look-up processes into parallel processes by means of the channel processor enables the overall throughput to be maximized. The linking of registers, on the other hand, eliminates the need for time consuming scans of all registers matched in the first compare cycle and allows full utilization of the associative nature of the processor. In the absence of these two features operating in conjunction with one another, there would be very little difference in operating speed between a conventional processor and one of an associative nature when the basic cycle tine was the same.

TABLE I

Dictionary		
Number of Words		44821
COMDICT	28409	
Other	16412	
Text		
Number of Words		4.0×10^6
Dictionary	3.9×10^6	
Non-Dictionary	1×10^6	
% Dictionary	97.35%	
Unique Words		71805
Dictionary	44821	
Non-Dictionary	26984	
% Dictionary	62.4%	
Word Length		
Unique		8.53
σ		3.61
Total		5.21
σ		2.09
New Terms		160/240,000 Total Words
Special Dictionaries		
High Frequency		
Number of Words		120
% of Total Text		45.6%
Number of Total Text		1.82×10^6

TABLE II. BINARY SCAN

8000 Words/Unit

Merged Dictionary

Words Compared	8,000
Found	7,788
Not Found	212
Total Compares - Found	117,416
0-7 Characters	47,445
8-14 Characters*	64,249
15-21 Characters	5,478
Over 21 Characters	244
Total Compares - Not Found	3,392
0-7 Characters	1,375
8-14 Characters	140
15-21 Characters	18
Total Compares	120,808
Average Compares per Input Word	15.101

*Assumes complete compare before decision.

TABLE III. STANDARD LOOK-UP (TWO-STAGE BINARY SCAN)

8000 Words/Unit

High Frequency Dictionary	
Words Compared	3,372
Found	3,280
Not Found	92
Total Compares	19,031
Found	18,512
Not Found	519
Words Not Compared	4,628
Total Pseudo Compares	4,628
Total Compares (h.f.)	23,659
Standard Dictionary	
Words Compared	4,720
Found	4,508
Not Found	212
Total Compares - Found	67,967
0-7 Characters	27,458
8-14 Characters*	37,177
15-21 Characters	3,175
Over 21 Characters	157
Total Compares - Not Found	3,402
0-7 Characters	1,437
8-14 Characters	1,842
15-21 Characters	105
Over 21 Characters	18
Total Compares (s.)	71,369
Total Compares Overall	95,028
Average Compares per Input Word	11.879

*Assumes complete compare before decision.

TABLE IV. MODIFIED LOOK-UP

8000 Words/Unit

High Frequency Dictionary

Words Compared	3,372
Found	3,280
Not Found	92
Total Compares	6,644
Found	6,560
Not Found	184
Words Not Compared	4,628
Total Pseudo Compares	4,628
Total Compares (h.f.)	11,272

Standard Dictionary

Words Compared	4,720
Found	4,508
Not Found	212
Total Compares - Found	67,967
0-7 Characters	27,458
8-14 Characters*	37,177
15-21 Characters	3,175
Over 21 Characters	157
Total Compares - Not Found	3,402
0-7 Characters	1,437
8-14 Characters	1,842
15-21 Characters	105
Over 21 Characters	18
Total Compares (s.)	71,369
Total Compares Overall	82,641
Average Compares per Input Word	10.033

*Assumes complete compare before decision.

TABLE V. RANDOM CHAIN SCAN

8000 Words/Unit

High Frequency Dictionary

Words Compared	3,372
Found	3,280
Not Found	92
Total Compares	6,644
Found	6,560
Not Found	184
Words Not Compared	4,628
Total Pseudo Compares	4,628
Total Compares (h.f.)	11,272

Standard Dictionary

Words Compared	4,720
Found	4,508
Not Found	212
Total Compares - Found	23,907
0-7 Characters	9,310
8-14 Characters*	13,195
15-21 Characters	1,331
Over 21 Characters	71
Total Compares - Not Found	1,378
0-7 Characters	549
8-14 Characters	773
15-21 Characters	47
Over 21 Characters	9
Total Compares (s.)	25,285
Total Compares Overall	36,557
Average Compares per Input Word	4.570

*Assumes complete compare before decision.

TABLE VI. ASSOCIATIVE SCAN (SINGLE MEMORY)

8000 Words/Unit

Merged Dictionary

Words Compared	8,000
Found	7,788
Not Found	212
Total Compares - Found	12,654
0-7 Characters	3,283
8-14 Characters*	8,316
15-21 Characters	999
Over 21 Characters	56
Total Compares - Not Found	343
0-7 Characters	89
8-14 Characters	226
15-21 Characters	24
Over 21 Characters	4
Total Compares	12,997
Average Compares per Input Word	1,625

* Assumes complete compare before decision.

TABLE VII. ASSOCIATIVE SCAN (DUAL MEMORY)

8000 Words/Unit	
High Frequency Dictionary	
Words Compared	3,372
Found	3,280
Not Found	92
Words Not Compared	4,628
Standard Dictionary	
Words Compared	4,720
Found	4,508
Not Found	212
Total Compares - Found	7,327
0-7 Characters*	1,900
8-14 Characters	4,812
15-21 Characters	579
Over 21 Characters	36
Total Compares - Not Found	339
0-7 Characters	93
8-14 Characters	224
15-21 Characters	18
Over 21 Characters	4
Total Compares (s.)	7,666
Average Compares per Input Word	.958

*Assumes complete compare before decision.

SECTION IV

TEXT STATISTICS

PROBLEM STATEMENT

The text statistics problem encompasses generating the data required to describe statistically the characteristics of a body of text. These statistics are used in various studies to determine the optimum manner of manipulating the text flow in certain data handling situations. For example, assume that one is investigating a sample of text from a data base which will eventually be processed automatically on a repetitive basis (e.g. daily intelligence reports). Then one is interested in acquiring data on numbers of words, word length distribution, etc., to be used in organizing a dictionary containing words in the material and information on processing to be associated with them, determining the rate at which new words will be encountered, investigating the volume and type of errors found, noting peculiarities of spelling or abbreviation, and identifying any other anomalies which might affect subsequent processing.

The final data to be derived from the text analysis should include the following items:

1. Word Frequency Counts
    a. Alphabetic sort with word frequencies
    b. Length of word sort with word frequencies
    c. Frequency sort, subsorted alphabetically with word frequencies

2. Growth Rate Information

   a. Word list in the sequence encountered

   b. Growth rate of new words per unit of text processed

3. Sorted subsets of words

   a. All alphabetic characters

   b. All numeric characters

   c. Only alphabetic and hyphen characters i.e. Franco-Prussian, Pierre-Laval, etc.

   d. Alphabetic, numeric and hyphen mixtures i.e. B-29, F-111, etc.

   e. Alphabetic and numeric mixtures i.e. 19H, 1604A, etc.

   f. All items not subsumed above.

Since all of these outputs are derivable from the basic frequency count of the words, the design and study effort was directed to consideration of the solution of this problem in an optimum manner.

It cannot be assumed that statistics derived from one body of text necessarily are transferable in their entirety to another body of text. Stylized vocabulary and technical terms tend to make the statistics descriptive of the particular data sample and can even be used to assist in determining to what body of text an unknown document or text sample belongs. Therefore the problem of determining text statistics is not one which can be accomplished once and then never repeated, but is likely to recur even within one organization where problems of interest change with time.

The data input to the text statistics processor is considered to be primarily textual. It might be unformatted text from teletype lines or from teletypesetter input. It will, of course, not be limited to these and can include a wide range

of other material. A primary characteristic of the data is that the text words appear as strings of characters with the word boundaries not specifically identified. The following character configurations are frequently the ones to be considered in defining the boundaries of text "words." The list can be modified to meet the requirements of the data sample in any particular instance by the addition of special terms or the deletion of configurations that are not meaningful in the context of a particular problem.

a. 'b̸' , the character 'blank'

b. ',b̸' , the pair ' comma blank'

c. '.b̸', the pair ' period blank'

d. ';' , the character 'semi-colon'

e. ';b̸' , the pair 'semi-colon blank'

f. ' :' , the character 'colon'

g. ' ;b̸' , the pair 'colon blank'

h. ' "b̸', the pair 'quotes blank'

Special characters other than shown may also be included in the usage given for colon, semi-colon, quotes, etc. Other usage may be defined as the occasion demands.

There is generally no special control information to separate text units as they are received, for example, on teletype or optical character or reader input. The separation is provided in the spacing of the information, which is adequate for production of printed output but poses processing problems for a computer. Thus the processor will need to be able to do part of the input editing task as well as the frequency counts and sortings. This requirement is made

necessary by the wide range of material for which this data is valuable and for which these general properties hold. Examples of this data are the text of news publications, intelligence reports, newswire data and other publication data as it appears prior to the printing.

CHARACTERISTICS OF THE PROBLEM

The text statistics problem may be simply stated as the need to generate lists of character strings of certain characteristics from an input string of text. Appended to each string will be frequency of occurrence information as well as the location in the text at which the string first appeared. Ordinarily the strings desired are words in the dictionary sense but no such restriction can be imposed in a limitation on the problem since mis-spellings and other errors are of considerable interest to the researcher. The terms 'words' and 'strings' or 'text items' are used interchangeably to describe the output required.

The initial sorting of the words is most easily accomplished as an alphabetic sort with the frequency and word length sorts to follow at a later time. This minimizes the data flow in the system by compressing the data to its final size as early as possible. The problem then, to be studied in relation to the text statistics processor will be limited to the generation of word frequency lists sorted alphabetically. The remaining lists can be generated as a table from this basic list by common sorting procedures, which are known to be input-output limited Thus, there will be little difference in the execution times between the associative processor and a conventional processor. The relation between these two philosophies will, of course, be affected by the additional burden imposed

by an equal load for each design. An equal load imposed upon each of two systems which differ in time of execution for the problem under consideration will have an effect in the execution time for the overall process that may range from a marked difference to almost none at all. Assume that for a problem in question that one process represents 80 percent of the job and that the remaining 20 percent of the job has been evaluated for time of execution. Let processor A have an execution time of 100 time units for the 20 percent part of the job while processor B does the same part in 150 time units. Then, if both have the same time of execution for the 80 percent portion of the job, in this case 400 times units if we use the time of processor A as the bench mark for the 20-80 split, processor A will complete the job in 500 time units while processor B will complete the same job in 550 time units. The difference in time of execution is still the same, 50 time units, but we now refer to the time of execution of A as being 90.5 percent of that of B rather than the 66.7 percent previously thought when considering only the small portion of the job. Careful consideration must be given to the problem of the overall execution for the problem by not limiting the investigation to the initial problem of the frequency-alphabetic sort.

Four schemes of data handling to solve the text statistics problem were proposed for use on a conventional design as well as on an associative design. These were based on the following assumptions:

1. **Capacity of the main store of both the conventional and the associative processors is insufficient to contain all unique entries.**

2. The input/output rate of each processor is assumed to be equal to its basic compare cycle.

3. The output of the data analysis is as previously specified.

The four methods of handling the data are:

METHOD I. Break the input text into "words" and generate a dictionary of unique items until storage capacity is reached. Save on a temporary medium any items not appearing in the dictionary after this time. Positional values for the first occurrence of each word in the dictionary are retained in memory. They are also associated with each of the overflow items saved on intermediate storage. At the end of the original input, the dictionary is saved and the overflow entries are re-entered to form additional dictionaries in the same manner. The additional dictionaries are merged with the original one and formed into the final alphabetic output.

METHOD II. Break the text into "words" and assign sequence values for input. Generate a dictionary in memory until storage is filled. Save the current dictionary on temporary media and begin the generation of a new dictionary. Continue in this manner until all of the input is exhausted. Merge the intermediate dictionaries deleting the extraneous sequence values.

METHOD III. Break the text into "words", assign sequence values and generate a dictionary of unique items until storage is filled. Write out onto temporary storage those items whose

frequencies to this point are below some threshold value. Continue to take input to regenerate the dictionary in the available core. The threshold is revised periodically to insure that some minimal percentage of memory is available for this function. Merge the resulting dictionaries retaining lowest sequence values and eliminating duplicate "word" items.

METHOD IV. Break the text into "words" and assign sequence values. Match these words against a pre-stored dictionary making the appropriate notations with each dictionary entry. Augment the pre-stored dictionary with any "new" term found until storage is filled. At this time all cells with zero frequencies are purged and the process continued until storage is again filled. The "new" terms are flagged and purged and saved at each time the dictionary reaches memory capacity. At the end of the input, the saved dictionaries are merged, duplicate entries are eliminated and frequency counts as well as sequence values are updated.

PROCESSOR DESIGN

The text statistics processor design is based on the assumption that there can be but one data input to the processor for any one problem. Thus, the processor will be input/output bound at some I/O speed based on the scheme for processing the text. This means that the processor must operate faster if an overall increase in throughput speed is to be accomplished.

The design is partitioned into an I/O system, a system control unit and a memory system.[1] The I/O system consists of the I/O units and the associated controllers, channel devices, etc. The memory system consists of two associative memories and the mask-compare registers and control circuitry needed by them. The system control exercises control over the I/O and all internal data flow in the system. It is responsible for retreiving and interpretation of all instructions of the system.

The two associative memory units of this system are of radically different sizes; however, the functions are quite similar and overlap to a large extent. The larger of the memories has registers capable of storing approximately 25,000 text words and their frequencies. The mask-compare registers of this memory are of the same length as the memory registers and allow for the inhibiting of compares on any specified subset of the word. The arithmetic-logic unit of this memory is used primarily in the detection of single matches and in the updating of counts of the words as the data is passed through the system.

The smaller of the memories is a special memory used for the function of word breakup and special character recognition. The memory is large enough to contain each possible character of the alphabet as well as an additional list of short, high frequency words. The alphabetic characters are used in the word boundary algorithm so that the boundaries of the incoming words are recognized.

---

[1] See Figure 7

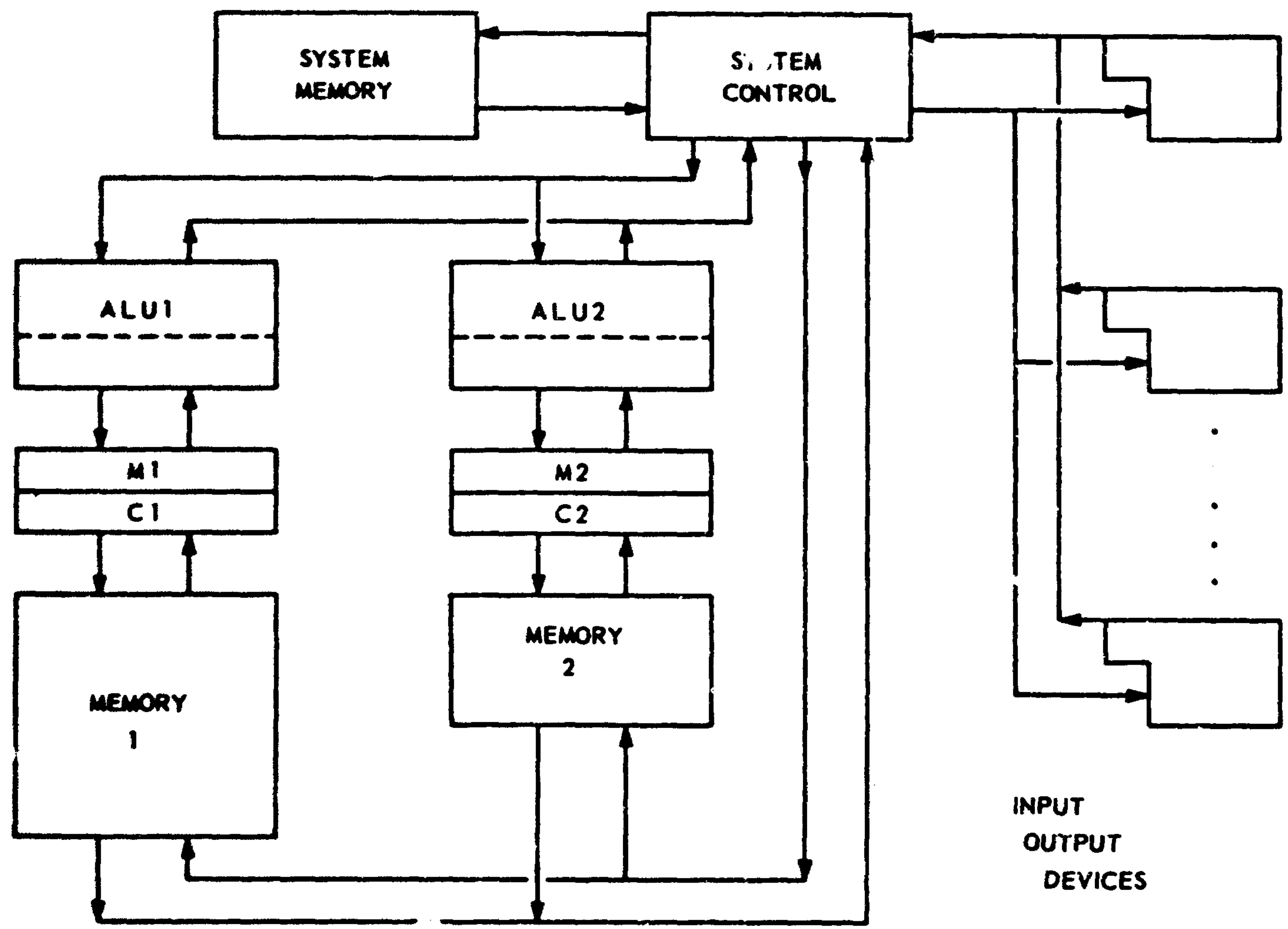


Figure 7. Processor for Test Statistics

This process can be accomplished in the time between transmission of the indi-vidual characters of the data stream.

Another function of this memory, apart from that already mentioned, is to eliminate from the main memory scan those words whose frequency is extremely high relative to the data set. By this means, it is possible to save the time in processing that would have been taken up by the search for those words. This memory unit is assumed to be of such an operating speed to be invisible in the data stream. This means that the search in the unit is done between characters and constitute no loss to the system. By this mechanism it is possible to match and count the frequencies of about 45 percent of the raw text words. This reduction in the flow of data arriving in the large memory means an almost 50 percent increase in operating speed.

The operations of the two associative memories are quite basic in nature and consist of the usual compare for exact match (with the match for bit selection), read out first match, write in first location, multi-write and link between registers. The link between registers, operates by setting a bit in the next register in memory from a register for which the compare condition is met. That is, for any register in memory that matches the associative search criteria and normally would have been marked, the register adjacent next higher in memory will now be marked. Read out is made on the basis of continuing exact matches so that only the last register of a sequence of registers matched by the search program will be marked in the case of the single match. This result occurs since the memory contains only one occurrence of each of the dictionary terms. Furthermore, it is of value to have only the last register

matched with the match bit set since statistics are being generated about each term and it is convienient for this information to be stored after the term in memory. It may be stored in the last register or in the one following, depending upon the number of characters involved. In a multiple match condition, of course, more than one of the memory registers could be marked. Match bits are assumed to be reset by the initiation of the write match bit portion of a succeeding match cycle. This is necessary in order to enable the search program to find those words which are more than one register in length and to mark only the last memory register for each of these words. Linking is assumed to progress one way only in this computer design, although the general case is to allow this type of function to be symmetric for searches in either direction.

The register length is such that a seven character string may be contained in each register. This choice was made based upon the reasonableness for hardware requirements and the distribution of the word length in the data sample. It is assumed in this case that some finite number of characters must be specified and for any particular length of register, there will be a requirement for a register of length longer than that. (See table III.) In addition to the seven characters per register, there is space for the counts associated with the lower frequency (1000 or less) words in each register plus some control information to indicate the relative position of the character string in the word.

EXAMPLE PROBLEM

A text sample and a dictionary derived from it were defined. The text was taken to have similar characteristics to those of a large sample of text 4.6

million words of text approximately. This represented approximately 144,000 unique words in text with the average frequency of occurrence for each word being about 94. The range of word frequencies was 1 - 264,491. These words were compared to a dictionary/[1] derived from Webster's Collegiate Dictionary. The words in the dictionary did not include any proper nouns or technical terms; that is, the set consisted of only common nouns, verbs, adjectives, etc. Among this restricted set, there appeared 28,409 words out of the 71,805 correctly spelled unique english words of text. The remaining 72,000 words were alphamerix mixtures, error, etc. The distribution of the lengths of the text words was such that the average length for all words of the sample was 5.21 characters while the average length for the unique words only was 8.53 characters. This reflects the high proportion of words such as 'an', 'the', 'for' and so forth. These words dilute the overall distribution of the lengths and skew it to the right. (See Table VIII.1a and VIII.1b.)

The input text to the processor is read one character at a time. The input is such that the possibility of parallel inputs is excluded by the requirement that the order of appearance of the unique words be preserved. The words and their boundaries must be identified and the word compared against a dictionary that is being constructed on the fly while the text is passing. Overflow occurs when the main storage is filled and a word enters the system for which no compare exists in the current dictionary. Such words must be saved together with a

---

[1] Comdict: A Common English Word Dictionary Private communication from M. Jones June 1963

notation of their sequence information until such time as the initial pass at the material is completed. They will then be passed again against a new dictionary and the cycle repeated until all of the words of the sample have been counted and the relevant information recorded.[1]

Each of the methods listed under CHARACTERISTICS OF THE PROBLEM was evaluated by means of the data model. The results are summarized in Tables XI through XIV. Table XI lists in detail the comparison times used in the standard random access processor as the data is processed through, in this case, five sorting passes and a merge pass. For each pass, the number of words read into the computer is listed as well as the total number of comparisons times utilized in the sorting. A random chain sort is assumed with an average compare time of 4.57 compares per input word. Table XII summarizes similar information about the associative processor design. The figure for the search time in this case is .958 compares per input word. The low (4.57) figure obtained for the random access processor is based upon the assumption of an average of 1.9 compares[2] per input word after the generation of the random memory address and 3.0 compare times for the address generation. In the case that the algorithm is not as efficient in producing the memory address, the corresponding compare times figure will have to be revised upwards.

---

1 Table IX and X give additional information of a statistical nature concerning the characteristics of the words in the test sample.

2 Dictionary Techniques
J. B. Ratchford, 31 May 1962
AIDf/SAC Subsystem Working Paper

Table XIII contains the information relative to the process which occur after the initial alphabetic sort has been made. The tasks here, with the exception of the frequency sort and the sequence encountered information, are performed in essentially the same amount of time on the associative processor and the random access processor. The major factor here is the I/O time and not the internal look-up time.

These results in Table XIII are obtained by assuming that the time of output and of re-input are equal to the compare time for the machine involved. That is, the time to read or write one 'word' in the I/O process is equal to the time necessary to accomplish one compare. This time is essential to the computations in that it is the main discriminator between the schemes for handling the data. Table XIII is the common information for those parts of the process in which the associative store is taken to yield no advantage.

Table XIV is a summary of the over processing time for the 4.0 million words approximately of text in the data model. Total comparison times are listed for each machine configuration as well as for each of the processing methods involved.

DISCUSSION

The overall result of the processor study is that the application of an associative processor in this area may yield a return of about three to one o[illegible]d. The cause for this low return is that the major portion of the task is linear and not conducive to parallel processing. The sorting of items in ordered lists, merging of lists is essentially I/O bound on present day computers

and appears as if it will remain so for the forseeable future. The gain for initial sorting of the text words is approximately four to one for similar reasons. Modern programming techniques enable the properties of associativity to be available on random access computers by such means of hash addressing, chain sorting and binary scans.

The main features of this design are the separation of the word break task into what amounts to a channel device from the central processor and the linking of adjacent registers in the associative memory during the scan operation for dynamic utilization of the interim results of the scan. Without these two features, it would be difficult to display a significant improvement in operating speed over the time of execution in a conventional processor.

TABLE VIII. 1a. TEXT STATISTICS DATA DESCRIPTION

I. TOKENS			4.0×10^6
A. ALPHABETIC		3.88×10^6	
1. Common Words	3.3×10^6		
a) frequency ≥ 30	2.9×10^6		
b) frequency < 30	4.4×10^5		
2. Other Words	5.2×10^5		
a) frequency ≥ 30	4.1×10^5		
b) frequency < 30	1.1×10^5		
B. NUMERIC		8.7×10^4	
C. ALPHA-HYPHEN		3.1×10^4	
D. ALPHA-NUMERIC		1.3×10^3	
E. ALPHA-NUMERIC-HYPHEN		1.2×10^4	
F. REMAINDER		3.0×10^4	
II. TYPES			1.4×10^5
A. ALPHABETIC		1.1×10^5	
1. Common Words	3.0×10^4		
a) frequency ≥ 30	1.4×10^4		
b) frequency < 30	1.6×10^4		
2. Other Words	7.7×10^4		
a) frequency ≥ 30	2.5×10^3		
b) frequency < 30	7.3×10^3		

TABLE VIII. 1a. TEXT STATISTICS DATA DESCRIPTION (Cont'd)

B.	NUMERIC	6.2×10^3
C.	ALPHA-HYPHEN	1.6×10^4
D.	ALPHA-NUMERIC	3.2×10^3
E.	ALPHA-NUMERIC-HYPHEN	2.1×10^3
F.	REMAINDER	6.1×10^3

III. MISCELLANEOUS INFORMATION

A.	Highest Frequency 20,000 words	= 95% of total words
B.	Average Frequency for words ≤ 30	= 5.62
C.	Unique Words frequency ≤ 30	= 124,000 words
D.	Total Words frequency ≤ 30	= 800,000 words

TABLE VIII. 1b. TEXT STATISTICS (FREQUENCY HISTOGRAM)

FREQUENCY	TYPES	TOKENS
1	63090	51090
2	19720	29440
3	10238	22614
4	6634	19736
5	4819	18095
6	3868	17508
7	3187	16709
8	2528	15344
9	2118	14272
10	1779	13390
11	1581	13101
12	1329	12948
13	1068	12584
14	798	11172
15	802	12030
16	695	11120
17	619	10523
18	551	9918
19	546	10374
20	482	9640
21	432	9072
22	405	8912
23	395	9085
24	342	8208
25	347	8675
26 AND GREATER	11718	

TABLE VIII. 1b. TEXT STATISTICS (FREQUENCY HISTOGRAM) (Cont'd)

FREQUENCY	TYPES	TOKENS
26	313	8138
7	306	8262
8	300	8400
9	285	8265
30	240	7200
1	237	7347
2	224	7168
3	236	7788
4	194	6596
5	202	7070
6	199	7164
7	169	6253
8	153	5814
9	153	5967
40	174	8960
1	158	6478
2	129	5418
3	134	5762
4	153	6732
5	136	6120
6	119	5474
7	129	6063
8	127	6096
9	101	4949
50	105	5250
51 AND GREATER	7042	

TABLE IX. FREQUENCY DISTRIBUTION OF DATA SAMPLE WORDS BY WORD LENGTH

LENGTH	FREQUENCY	LENGTH	FREQUENCY
1	14	13	1628
2	76	14	1059
3	813	15	694
4	1962	16	451
5	3420	17	291
6	5867	18	203
7	6741	19	123
8	6159	20	89
9	5277	21	64
10	4249	22	42
11	3208	23	30
12	2348	24 + GREATER	13

TABLE X. FREQUENCY DISTRIBUTION OF DATA SAMPLE WORDS BY FIRST CHARACTER

CHARACTER	FREQUENCY	CHARACTER	FREQUENCY
A	2505	N	1782
B	2908	O	1181
C	3212	P	2764
D	1825	Q	138
E	1667	R	1857
F	2110	S	4818
G	1646	T	2342
H	1983	U	944
I	3420	V	675
J	579	W	1329
K	859	X	13
L	1442	Y	214
M	2390	Z	212

TABLE XI. NUMBER OF COMPARISON TIMES FOR THE ALPHABETIC SORT USING RANDOM ACCESS PROCESSOR

			Number of Compares
I METHOD I			
A. Pass I			
	1. Words Read	4,001,562	
	2. Words Saved	24,921	
	3. Overflow	217,343	18,638,074
B. Pass II			
	1. Words Read	217,343	
	2. Words Saved	24,747	
	3. Overflow	95,378	1,161,072
C. Pass III			
	1. Words Read	95,378	
	2. Words Saved	25,070	
	3. Overflow	45,376	529,011
D. Pass IV			
	1. Words Read	45,376	
	2. Words Saved	24,307	
	3. Overflow	17,591	258,062
E. Pass V			
	1. Words Read	17,591	
	2. Words Saved	- 0 -	
	3. Overflow	- 0 -	80,391
F. Merge			263,421
G. Total Compares			20,930,031
II METHOD II			
A. Pass I to Pass XV			
	1. Words Read	240,000	
	2. Words Saved	24,921	17,012,725
B. Pass XVI			
	1. Words Read	161,562	
	2. Words Saved	- 0 -	738,338
C. Merge			
	1. Pass I		483,515
	2. Pass II		259,687
D. Total Compares			18,493,765

TABLE XI. NUMBER OF COMPARISON TIMES FOR THE ALPHABETIC SORT USING RANDOM ACCESS PROCESSOR (Cont'd)

			Number of Compares
III	METHOD III		
	A. Pass I to Pass XV		
	1. Words Read	240,000	
	2. Words Saved	4,830	16,560,675
	B. Pass XVI		
	1. Words Read	161,562	
	2. Words Saved	- 0 -	738,338
	C. Merge		
	1. Pass I		182,150
	2. Pass II		259,687
	D. Total Compares		17,740,085
IV	METHOD IV		
	A. Pass I to Pass		
	1. Words Read		
	2. Words Saved		16,564,500
	B. Pass		
	1. Words Read		
	2. Words Saved		738,338
	C. Merge		
	1. Pass I		184,400
	2. Pass II		259,687
	D. Total Compares		17,746,925

TABLE XII. NUMBER OF COMPARISON TIMES FOR THE ALPHABETIC SORT USING ASSOCIATIVE ACCESS PROCESSOR

I METHOD I			Number of Compares
A. Pass I			
	1. Words Read	4,001,562	
	2. Words Saved	24,921	
	3. Overflow	217,343	4,184,432
B. Pass II			
	1. Words Read	217,343	
	2. Words Saved	24,747	
	3. Overflow	95,378	376,029
C. Pass III			
	1. Words Read	95,378	
	2. Words Saved	25,070	
	3. Overflow	45,376	184,506
D. Pass IV			
	1. Words Read	45,376	
	2. Words Saved	24,307	
	3. Overflow	17,591	94,164
E. Pass V			
	1. Words Read	17,591	
	2. Words Saved	- 0 -	
	3. Overflow	- 0 -	16,852
F. Merge			170,132
G. Total Compares			5,026,179
II METHOD II			
A. Pass I to Pass XV			
	1. Words Read	240,000	
	2. Words Saved	24,921	3,486,182
B. Pass XVI			
	1. Words Read	161,562	
	2. Words Saved	- 0 -	154,776
C. Merge			
	1. Pass I		483,515
	2. Pass II		259,687
D. Total Compares			4,384,160

TABLE XII. NUMBER OF COMPARISON TIMES FOR THE ALPHABETIC SORT USING ASSOCIATIVE ACCESS PROCESSOR (Cont'd)

		Number of Compares
III METHOD III		
A. Pass I to Pass XV		
1. Words Read	240,000	
2. Words Saved	4,830	3,557,475
B. Pass XVI		
1. Words Read	161,562	
2. Words Saved	- 0 -	154,776
C. Merge		
1. Words Read		182,150
2. Words Saved		259,687
D. Total Compares		4,154,088
IV METHOD IV		
A. Pass I to Pass		
1. Words Read		
2. Words Saved		3,561,300
B. Pass		
1. Words Read		
2. Words Saved		154,776
C. Merge		
1. Pass I		184,400
2. Pass II		259,687
D. Total Compares		4,160,163

TABLE XIII. NUMBER OF COMPARISON TIMES FOR THE SORT FUNCTION PERFORMED SUBSEQUENT TO THE ALPHABETIC SORT

	Number of Compares
I LENGTH SORT: COMMON TO BOTH PROCESSORS	
A. Dictionary	170,132
B. Dictionary Output	113,421
C. Merge	283,553
D. Total Compares	567,106
II FREQUENCY SORT	
A. Standard Processor	
1. Dictionary Input	777,503
2. Dictionary Output	113,421
3. Merge	283,553
4. Total Compares	1,174,477
B. Associative Processor	
1. Dictionary Input	170,132
2. Dictionary Output	113,421
3. Merge	283,553
4. Total Compares	567,106
III DICTIONARY SCAN: COMMON TO BOTH PROCESSORS	170,132
IV SEQUENCE ENCOUNTERED	
A. Standard Processor	
1. Dictionary Input	777,503
2. Dictionary Output	113,421
3. Merge	283,553
4. Total Compares	1,174,477
B. ASSOCIATIVE PROCESSOR	
1. Dictionary Input	170,132
2. Dictionary Output	113,421
3. Merge	283,553
4. Total Compares	567,106

TABLE X[illegible]. NUMBER OF COMPARISON TIMES FOR THE SORT FUNCTION PERFORMED SUBSEQUENT TO THE ALPHABETIC SORT (Cont'd)

		Number of Compares
V	GROWTH RATE: COMMON TO BOTH PROCESSORS	- 0 -
VI	SUBSET DATA: COMMON TO BOTH PROCESSORS	
	A. Alpha	129,810
	B. Numeric	7,485
	C. Alpha-Hyphen	19,054
	D. Alpha-Digit	3,913
	E. Junk	9,867
	F. Total Compares	170,132

TABLE XIV. TOTAL COMPARISON TIMES BY PROCESSOR AND BY SORT METHOD

		Total Compares
I	STANDARD PROCESSOR	
	A. Method I	24,186,355
	B. Method II	21,750,089
	C. Method III	20,996,409
	D. Method IV	21,003,249
II	ASSOCIATIVE PROCESSOR	
	A. Method I	7,067,761
	B. Method II	6,425,742
	C. Method III	6,195,670
	D. Method IV	6,201,745

Section V

FORMATTED FILE PROCESSING

PROBLEM STATEMENT

In recent years there has been a rapid growth in the use of so-called "formatted file systems." These systems are general-purpose data storage, maintenance and retrieval systems designed to provide the user with a maximum amount of flexibility. They feature the use of a single set of programs to handle a variety of demands on a group of large files. Each file may possess a different format, but all records within a file must be identical in format. New files may be created or old files changed in format to meet new requirements. Data can be added to files, or changes can be made to correct errors in existing files. In each case, however, the prime requirement for these systems is the ability to make rapid responses to queries.

Each query consists of a logical combination of individual field name/field value pairs. For example, in a hypothetical file on automobile registrations, the license numbers of all black 1963 Fords might be requested. Specifically, the request would be: Print the contents of the License Number field for all records in the file with:

Color field contents = Black

AND

Date field contents = 1963

AND

Manufacturer field contents = Ford.

More complex logical statements and different types of relations (e.g., $\geq$, $<$) are of course permitted, and easy-to-use languages are usually provided for user convenience. However, the system allows the user to make any query involving the fields in any of the files using a set of logical and relational operators. Because it is impossible in most cases to predict the queries to be made, and because the files used are frequently large (as much as 75 tape reels of data for one intelligence file), the searching of such files presents a severe problem.

CHARACTERISTICS OF THE PROBLEM

The early formatted file systems used the one bulk storage medium available at the time, magnetic tape. Not much can be done in such a system to reduce access time. If one field is referenced in a large percentage of queries, the file may then be sorted and maintained in order on that field. Thus, one can index to the reel level and scan only to the point on that reel where the relevant records are located.

The advent of the quasi-random access, bulk storage devices such as disks, drums and strip files made more efficient file structures possible, since access is available at the track level, and scanning need only be performed within a track. Some work has been done on examining the general problem of file structure,[1] and specific examinations have been carried out for some critical files.[2]

---

[1] F. T. Baker, "Some Storage Organizations for Use with Disk Files," AIDS Working Paper, January 1963.

[2] J. B. Ratchford, "Notes on File Structure for AIDS Phase II System," September 1962.

The usual searching process is a relatively inefficient and slow means of locating data because of the tremendous amount of nonrelevant data in the file which must be scanned to find all relevant data. All records to be scanned must be transferred to a central processor and examined to some extent before being discarded if they do not meet the selection criteria. In some cases, the complete file may have to be scanned to ensure finding only a few records. This method is obviously wasteful, and some of the techniques described next have been developed to improve search efficiency.

In some cases, there is seldom a need for reading more than one record per transaction. In such cases, it is often possible to compute an address based on the value of the keys given. For instance, an algorithm may be desiged to operate on a randomly chosen set of keys provided to produce an evenly distributed set of addresses over the total storage area. When an address is computed, a scan will start at the compute address to find the actual record desired. In special cases involving one key, this has been very effective. However, it tends to order the file randomly, which naturally makes it impractical to find records for sequential or mass transfer.

Since the amount of data scanning during retrieval is roughly inversely proportional to the number of keys indexed, the usual indexed file will have several indexes. One file may require indexes only on its more important keys, another may require that all keys be indexed. If the indexes and files are ordered on the keys indexed, the index entries will address sets of records with the corresponding index valve rather than scattered single records. The query can then be accomplished by a search of an index to find the location of the first record of an ordered

set and then reading out the complete set. In such a file organization there is a good chance that one read will fetch several records at a time thus taking advantage of the mass transfer characteristics of the storage device.

Another common approach involves breaking up logical records into smaller groups of fields and then chaining the groups with cross-reference addresses to show relationships among various keys of the file. In some cases, this is a good compromise design that takes advantage of the mass transfer characteristics of the storage media, even when search may be on a variety of keys. Generally, when a scan is used it will be short.

In practice, search time is usually split among searching the index for the particular key value, seeking an area (e.g., a track or cylinder) where the record is kept, then scanning for the actual record or block of records within this area. The time spent on the first and last stages depends upon the level of indexing. When indexing is to the record level, no search of data file area is necessary. If indexing is to the track level, only a track will be scanned. Indexing can be to the cylinder level, requiring scanning of the cylinder tracks.

A fundamental rule in the generation of indexes is that no index should be used that gets closer to the data than required. For instance, if a disk system is used in track mode where the transfer of data requires the reading of a full track, indexing should be to the track level as opposed to the record level. Either way will find the record, but the second index will be longer than the first by a factor equal to the number of records contained on a track. This extra length adds to the index search time; more importantly, it greatly increases the index maintenance time during the update.

A common problem in many of these approaches is that, as indexing and chaining are used more and more, the time for adding records to a file increases significantly. The time spent in maintaining those features that facilitate query thus has a disastrous effect on overall system efficiency. This is especially critical in systems where input volume is high, as in many "history file" applications. Therefore, the application of sophisticated indexing techniques to facilitate query handling often proves to be self-defeating.

However, there is another approach that can reduce maintenance time without losing query efficiency. A faster scan over a larger area will allow smaller, courser indexes with a corresponding decrease in index maintenance time and can be accomplished with a parallel scan of many tracks. For instance, the cylinder concept common in disk operations can be modified so that all heads are read in parallel. With such a scheme, small indexes could be used to index large files for quick access. A further saving with this approach would be in the lessened need for indexes on seldom-used keys. A high-speed scan on such keys would be far preferable to the extra effort required for updating a seldom-used index for every update transaction. Also, queries are sometimes made on keys not even indexed, and high-speed scan would be invaluable in such cases. Finally, chaining could be simplified if such a scheme were adopted and were feasible. An attempt would be made to always cluster chained records on a cylinder, and each chain scan could be shortened to the extent that this clustering was successful.

The concept appears attractive, but has a major drawback; increased demands will be put on the central processor. With data transferring into the central processor in parallel from disks (for instance, at a 40 × 90 KC = 3600 KC

rate), the memory would soon be overwhelmed. Even with slower data rates, buffer space would be used, thus degrading computing performance of the system. The solution to this problem appears when the distribution of the work required in a scan is considered. The work involved in processing a file record can be broken down into four simple functions:

1. Get the records under the heads
2. Transfer the record to memory
3. Test the record for relevancy
4. Process for output or reject the record.

Of these, the first two generally require the greatest time to the extent that the other functions can often be overlapped. In the computer, the test for relevancy is usually simple. Basic logical operations on and between file items are sufficient to isolate the required records in the majority of cases. On individual fields, a high-low-equal compare operation is generally all that is required; and between fields the "and," "or," and "not" logical operations will usually serve adequately. Though simple, these operations nevertheless require a major portion of CPU time with relatively small dividends in terms of relevant records. In the quest for an upgraded CPU operation, this seems like a likely place to start.

It appears worthwhile, therefore, to investigate putting the scan and test functions on a "channel" basis. That is, the scanning and testing should be separate and distinct from the CPU; the "channel" should be set up at infrequent intervals and allowed to do its own searching. This appears to merely transfer the tremendous buffer out of the CPU and put it in the channel, but even this problem

can be attacked by allowing the records to stay on the storage medium until after they have been scanned and found relevant. This procedure can be done either by employing separate heads for scanning and for data transfer, or by allowing two revolutions for each transfer—the first for scan, the second for transfer, and both from the same heads.

One characteristic of a formatted file is that the records are all exactly alike in format in any single file, and can be stored in parallel and therefore scanned in parallel. Such a scheme could also allow a simple implementation of multi-query capability. In such a system, many comparison criteria at once from several queries could be compared to a single field from many records, multiplying the effective scan rate by the number of different parameters compared to a field.

Such a system would seem to have many advantages. Its autonomous nature should free the CPU for upgraded work. Because the system would be designed for both batched and single input handling, it should help keep the system running well under transient conditions. It should tend to considerably decrease the time of both update and query. Finally, its implementation appears to be presently feasible.

The design of such a parallel search channel is discussed in the next section. Since this processor offered promise of significant advantages, both in connection with General Purpose Associative Processor (described in Section VIII) and as a channel device to be used with conventional computers, it was decided to carry out this design in more detail than the design of the other problem area processors.

PROCESSOR DESIGN

The formatted file processor is designed for use with bulk storage devices where parallel data transfer from a number of identically formatted records is possible. Such devices are exemplified by head-per-surface disk files, head-per-track strip files, head-per-track drums and similar devices. (The processor will be desribed in terms of its application to a disk storage unit but is clearly applicable to the others.) This processor is also designed for use as an external device or channel processor from which data is transferred to a separate central processor for further manipulation.

The bulk store to be considered will consist of a disk unit with 40 data disk surfaces and two field control surfaces. Information is recorded on the disk surfaces in concentric tracks, each of which allows four thousand 8-bit characters to be recorded on it. There are 500 tracks on each surface.

Data and control information is read or written on these surfaces by 42 heads mounted on an access mechanism—that is, without moving, the access mechanism can read or write information on any one or all of 40 data tracks while being controlled by the corresponding format and timing tracks. The 42 tracks accessible without motion of the access arm are called a cylinder. A cylinder may be accessed at any time by a pair of mechanisms. One of the pair carries the read heads of the scanner, the other mechanism, assumed to be located 1000 characters further around the unit, carries both read and write heads.

All system operations are controlled by the format tracks. One of these tracks marks fields and records on the cylinder. The other provides timing, read/write synchronization, and access mechanism checking.

Data fields must be stored on the 40 tracks of a cylinder in parallel, that is, all records within the area defined by one record length along each of 40 tracks must be stored so that the records and fields within records begin an[illegible] the same angular position of the cylinder surface. A record length of 1000 characters or less is allowed.

Records with a variable number of fields are handled by breaking them up by group into fixed length records and chaining the various groups of a record together. In such cases, all equal-length groups will be stored in corresponding sections of the file. In a file with fixed groups (i.e., master records) and periodic groups (i.e., detail records in a different format than the master), the fixed groups would be stored together in parallel, and the periodics would be stored together in parallel in another section of the file.

Figure 8 shows a block diagram of the processor. As an example, the bulk store is shown filled with a set of records from a vehicle file. The bulk store represents cylinders of disk storage. The access mechanism is such that corresponding bits of characters under all heads are read into the Field Compare-Matrix (FCM) simultaneously. The bit read from each track is read into all positions of the column of the Field Compare Matrix immediately "above" the character. For example, the bits of characters C, T, B, C . . . . would be read into all positions of their respective columns in the matrix; following this, the bits of characters, A, R, U, A, ... would be read into all positions of the respective columns in the matrix and so on. This read-in actually is done parallel by bit—that is, one bit per character is read from each track simultaneously.

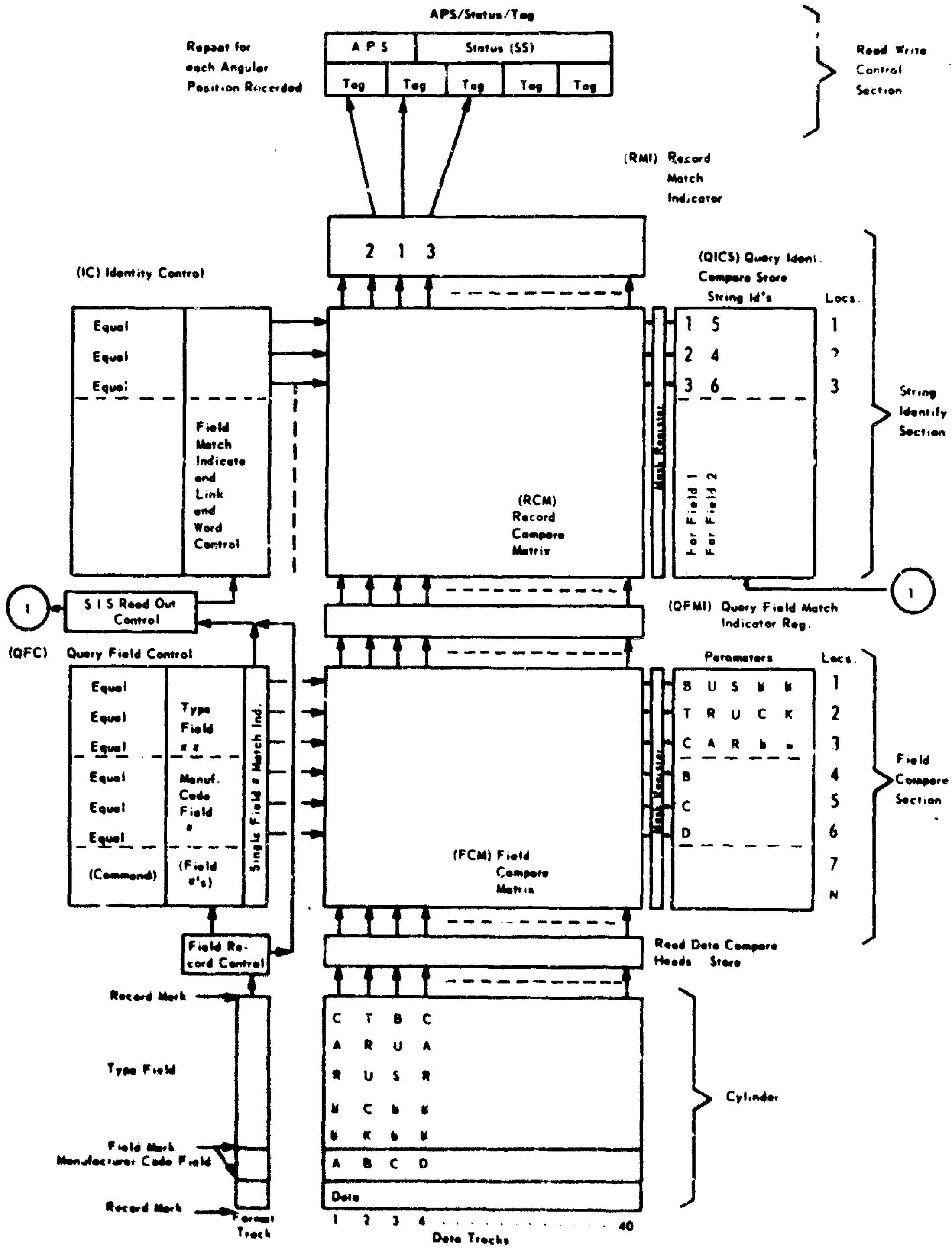


Figure 8. Associative Disk Scanner

Parameters for finding data are shown in the Query Data Compare Store (QDCS) and the Query Identification Compare Store (QICS), which are actually different sections of the same memory. During the loading process, the actual field contents to be compared are stored in the QDCS as shown. In addition, a string of FCM row addresses is generated indicating particular logical combinations of query data fields which will satisfy one or more queries. These addresses are stored in "characters" of the QICS; thus the row address of a QICS string will represent one query which will be satisfied by the logical conjunction of the several specified QDCS data fields. In Figure 3, for example, location one in QICS containing the FCM row addresses 1 and 5 represents the query for a "bus manufactured by C." QICS location three represents a query concerning a "car manufactured by D."

The Query Field Control controls the type of comparison which will be made on each field when that field is passing under the heads. In the example, when the Type field is passing under the heads an "equal" compare will be made. Likewise an "equal" compare is specified for the Manufacturer field.

The heart of the system is the Field Compare Matrix. This is a matrix of compare circuits, each match bit of which drives the readout of a single register of read-only storage. When a match bit is set, the associated register of read-only storage reads its own row address into the position of the Query Field Match Indicator directly above. Under control of the Query Field Control, the compare circuits match the contents of fields passing the scanner heads with those stored in rows of the Query Data Compare Store. If a field on a track matches a row of the Query Data Compare Store, the match bit at the intersections of the

corresponding row and column will be set. This in turn will read the address of the row which compared into the position of the Query Field Match Indicator corresponding to the track containing that field.

To understand the overall operation, consider the example shown in Figure 8. At the beginning of character time one, the first bit of a character from each of the set of characters C, T, B, C . . . . will be simultaneously read into all positions of the respective columns in the FCM. At the same time the mask register will be loaded, and the first bit-column of the Query Data Compare Store will be simultaneously compared to all columns of the FCM. Query Field Control (QFC) will only allow comparisons to be made on the Type field at this time.

Initially, all match bits are set off to indicate the no-match condition. Then the match bits of rows to be compared with the currently passing field are set "on" to indicate all matching condition. During actual comparison, the field which mismatches will cause a match bit to be turned off at the intersection of the rows and columns mismatched. Thus the result of the comparisons at the end of character time one would be match conditions shown in positions (1,3), (2,2), (3,1) and (3,4). Following the bits of the first row of characters, the bits of a second row, A, R, V, A, are read and a second set of compares are made with the bits of the second column of the Query Data Compare Store. Again, compares will be left in positions (1,3), (2,2), (3,1), (3,4). The same compares will clearly remain set when the end of the field is signified by the format track of the cylinder.

When the end of the field is signaled, a new set of actions will start. The QDCS row location numbers corresponding to match bits set will be transferred to the Query Field Match Indicator Register (QFMIR). Comparison of this set of location numbers against those in the first column of the Query Identification Compare Store (QICS) will be made in a manner analogous to that just described for the field data. This comparison will leave match bits set in positions (1,3), (2,2), (3,1) and (3,4) of the Record Compare Matrix (RCM).

In the meantime, the FCM will be reset and comparison of the second field will be carried out to find Manufacturer codes of B, C or D. This comparison results in match bits being set in FCM positions (4,2), (5,3) and (6,4). Again, at the end of the field, the QDCS location numbers corresponding to the match bits set will be transferred to the Query Field Match Indicator Register. The Record Compare Matrix will not be reset and comparison of the QFMIR with the second column of QICS will be made. This comparison will result in the match bits in position (3,4) being turned off, leaving only RCM positions (1,3), (2,2), and (3,4) still set. This condition will remain until the end of the record is reached, at which time QICS location numbers will be transferred by the match bit circuits to the Record Match Indicator Register (RMI). The location numbers now in the RMI indicate that, among the records which start at some pre-determined angular position of the disk, those tracks corresponding to the RMI position now filled have met query criteria. Also, the queries answered are identified by the QICS row numbers now in the RMI.

Actual data transfer is accomplished in one of two ways. The data may be transferred as single records by addressing a particular position on a particular

track as in the usual bulk store devices, or data may be transferred automatically under status control as follows.

Under control of the status register, the information is read by read/write heads separate from and following the scanner heads when the beginning of the record is encountered. The status registers tells which track to read and the tags identify a central processor address to which the data will be transferred. Writing is accomplished analogously.

Since this processor offers significant data handling advantages over conventional channel devices, it was determined that it would form an excellent auxiliary device for use with the General Purpose Associative Processor described in Section VIII. For this reason, the design effort was continued to provide further detail of its important component sections. The remaining part of this section is devoted to describing these components.

Field Compare Section

The Field Compare Section of the processor is primarily concerned with the parallel comparison of individual query fields against specified fields of the stored records. The important units of the Field Compare Section are the Field and Record Control, the Query Data Compare Store, the Field Compare Matrix, the Query Field Control and the Query Field Match Indicator.

Field and Record Control. The operation of the scanner as well as the read/write circuits is under control of the Field and Record Control (FRC). This control reads two tracks of control information. The first track (Field Marking Track) indicates the beginning of each field, as well as the record end. The

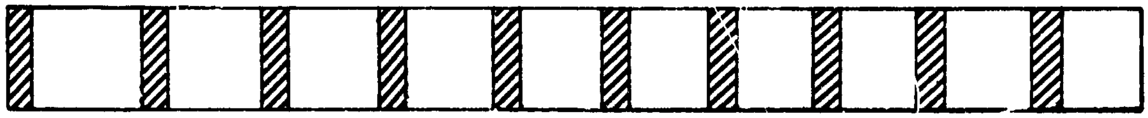
second track (Timing and Angular Position Track) indicates angular position and cylinder address. In addition, it provides a base timing pulse which, together with auxiliary timing circuits, allows for synchronized read and write.

The FRC reads the Field Marking Track and uses a field counter to indicate which field is being read. It senses the beginning-of-record mark which resets the field counter to one. Each change of field then increases the field counter by one. The count then is used to control the field comparisons to be performed by the identity and query field controls.

All read/write/scan operations are timed through use of information from the Timing and Angular Position Track (TAPT). Actual timing pulses are furnished by a combination of primary pulses from the TAPT and a clock. The clock is synchronized with the primary pulses. It is assumed that there is enough room on the TAPT to record angular position information interspersed with primary timing pulses at regular intervals around the track. With the angular position available as soon as the access mechanism settles to a new position, so-called rotational delay can be reduced. After access motion has stopped, it will be possible to start scanning as soon as the beginning of the first record is sensed following the first read of angular position. Thus, with several identically formatted records stored on a track, any delay will be only a small fraction of the rotational period.

Query Data Compare Store. The file is searched by comparing search terms stored in Query Data Compare Store (QDCS) against fields in all tracks of the cylinder being processed. Search terms are stored in the QDCS a word at a time as in a conventional memory, except that 10 bits are stored for each 8-bit

character of the word. The extra two bits per character are to indicate masking for each four bits of a packed numeric digit or for the eight bits of a character. Assuming a five-byte QDCS word, a word of digits stored would appear as follows:



The shaded part is the mask bit and the unshaded blocks each represent four bits of the data byte.

This memory is constructed so that input is by word while output (to the compare matrix) is parallel by bit. That is, the first bit of all words can be read in parallel, followed by the second bit of all words, etc. Every fifth readout, a new mask is read into the Mask Register. This mask will be used to mask the following four data bit comparisons, allowing for partial field comparisons. A one in a mask bit will prevent comparison on the next four bits; a zero enables comparison of the next four bits.

It is expected that this memory could be constructed in either of the following methods depending upon relative cost of components. If magnetic cores are used, the usual (XZ, YZ) coincident current write by word seems plausible, where the intersection of the XZ and YZ planes corresponds to word address. Output then could be accomplished through use of a full write pulse on the XY plane. Sense amplifiers on each Z axis would then sense switch cores throughout the XY plane. Because this method is expensive in terms of sense amplifiers, a better method might be to design the memory as a series of flip-flop registers.

<u>Field Compare Matrix.</u> The Field Compare Matrix is the heart of the Field Compare Section. It is comprised of a matrix of comparison and associated readout circuits to compare the row of bits, one from each track, presently being read from the cylinder with the column of corresponding bits presently being read from the comparison memory. The type of comparison is determined by the mask and the compare control. Types of comparisons are controlled by row and may be different for each row. In other words, a term in word one of the comparison memory may be equal—compared to a field at the same time that a term in the second word is high—compared to the same field. The only restriction is that the results of the two comparisons must be <u>mutually</u> <u>exclusive</u>, since, if more than one match bit is set in a column, the column will be treated as though all its match bits had been set.

Figure 9 shows the general comparison scheme used in the matrix. Only the operation of two query words on one track is shown. All other tracks operate in parallel as indicated. The mask bits from the first digit of all words of Query Data Compare Store (QDCS) are entered into the mask bit flip-flops for each word. Then the first data bit from each word is read into its corresponding compare flip flop. Here, a 1 is signified by C while a Ø is a $\overline{C}$. The first bit of the field being compared on each track is read by the corresponding disk head into all compare circuit in the corresponding column. These bits are compared if the mask bit allows it, and the results of the comparison are recorded in all comparison circuits of the corresponding rows. Similarly, successive rows of bits from the cylinder are compared to successive columns of bits from the compare memory with a new mask every fifth bit. At the end of the field, all matrix

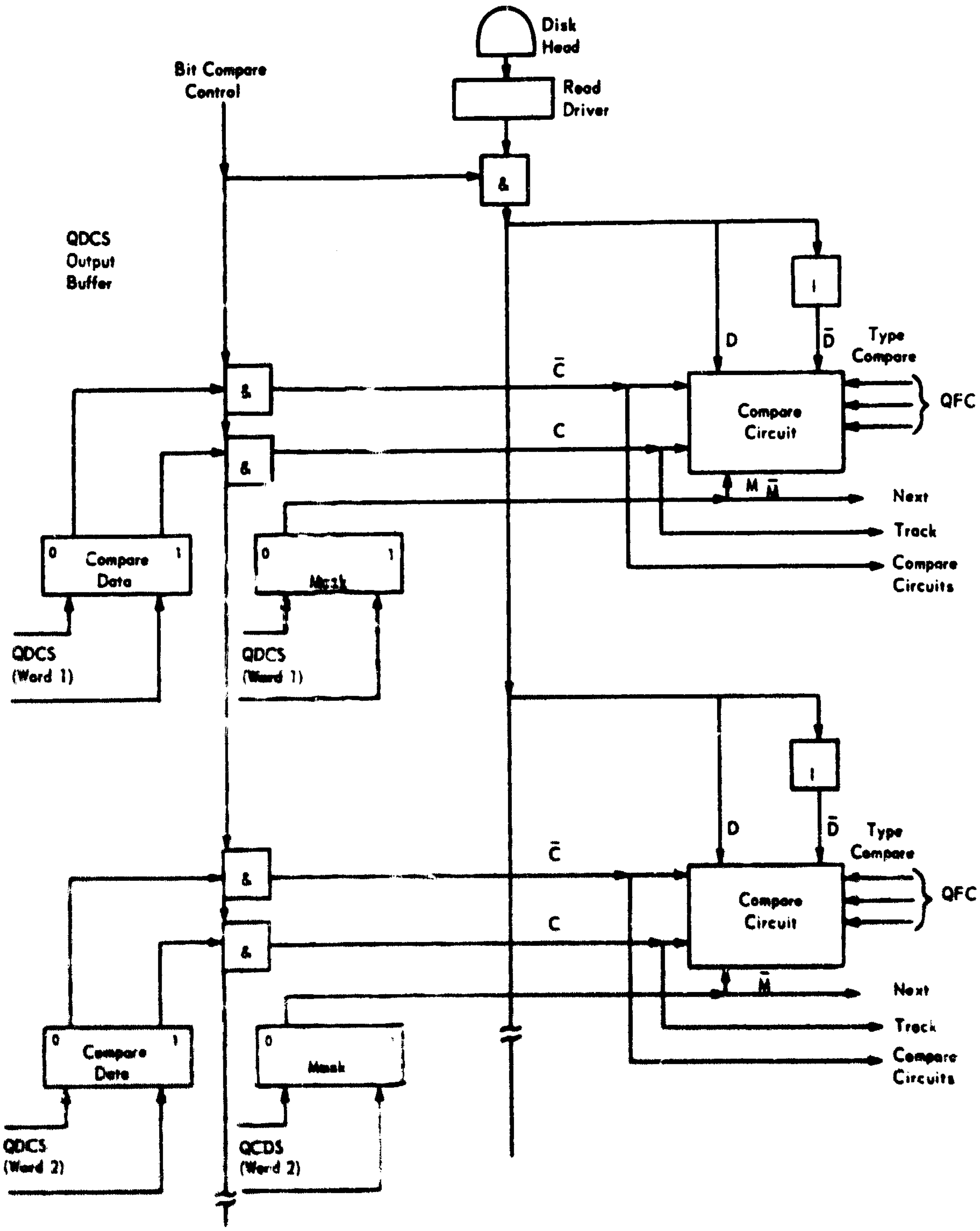


Figure 9. Parallel Read Compare by Bit

compare circuits are notified of the type of compare being attempted on that particular row by lines from the Query Field Control (QFC). Then the match bits are set to indicate the success or failure of the comparison of the field.

Figure 10 indicates the operation of the compare circuits of the matrix by detailing the compare circuit of the n^{th} row of one column. The circuit is designed for comparison of a sequence of bit pairs. High, low, and equal comparisons are possible in the sequence circuit. At the start of a field on the disk, all match bits in every row of the scanner are set to Ø for no-match; then the match bit flip-flops corresponding to the rows for the fields to be scanned are reset to 1, the match condition. The sequence change indicators, flip-flops A and B, are reset to Ø; then the comparison of compare bits and data bits proceeds as follows. An input of equal bits (CD or $\overline{C}\overline{D}$ will have no effect on the sequence change detection. The first unequal pair in the sequence will set one of the sequence detector flip-flops as follows:

$$A = \overline{C} D \overline{B}$$

$$B = C \overline{D} \overline{A}$$

From this it can be seen that once A is set (data high), B can no longer be affected by the sequence. Similarly, B set (data low) disables the setting of A. Thus, the change detector detects and records the direction of the first inequality of bit pairs. This enables the compare control lines to properly set the match bit to the no-match condition if necessary at the end of the field or word, whichever is shorter.

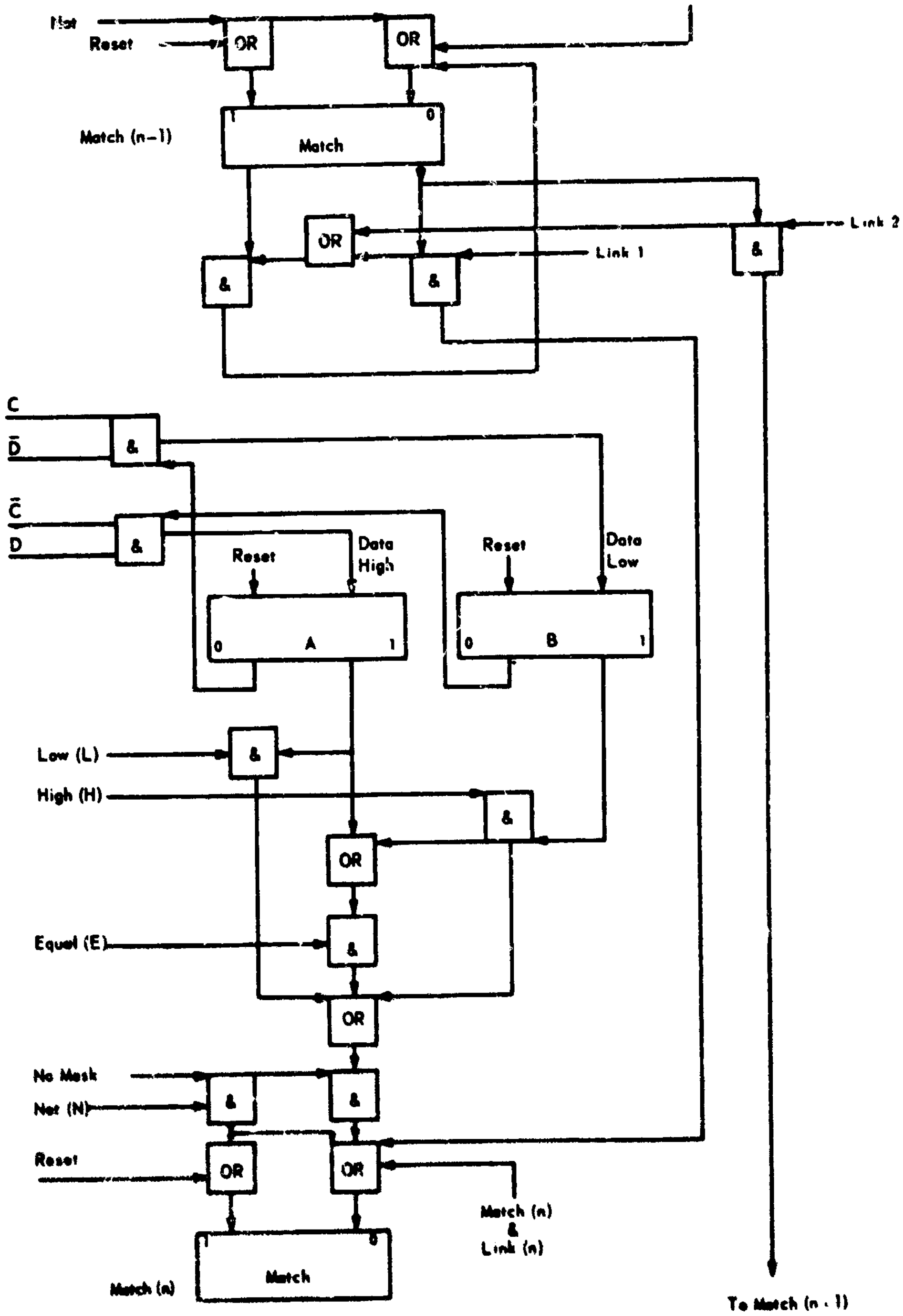


Figure 10. Field Compare Matrix Compare and Link Circuits

The action of the High, Low, Equal, Not and No Mask lines is obvious. Excluding the actions of the link circuits, the condition for "match" can be described by either

$$\overline{M} = (AL + BH + E(A + B))\ \text{No Mask} + NM$$

or

$$M = E\,\overline{A}\,\overline{B} + LB + HA + N(AL + BH + E(A + B)\ \text{No Mask}$$

The Not line in concert with No Mask merely complements the match bit after other actions have taken place.

When a field is longer than a word of the Query Data Compare (QDCS), the link mechanism must be used to link successive words of the field. Link 1 links successive words, for use when the field is so stored in the QDCS. One function to be performed by the scanner is the between limits search, for which the search limit parameters must be stored in alternating words of the QDCS. When these fields are more than one word long, Link 2 must be used to link successive words of each field.

Figure 10 shows the effect of the Link 1 line on the match bits. After all match bits have been set to their proper values at the end of a word, the Link 1 lines (under program control) will transfer the no-match condition from one match bit (Match n-1), for example, to the next one down (Match (n)). In addition, if Match (n-1) is in match condition it will be set to no-match by the Link 1 line. This ensures that at the end of the field, only one match bit can be set for that field. Link 2 required in the between limits search operates in a similar manner, except that the link is from one match bit to another two below the first—for example from Match (n-1) to Match (n + 1).

Once the match bits are set for a field, it becomes necessary to read out the information that the field on some track (or tracks) has matched the scan parameter in somerow. The read-out mechanism is a read-only memory, each word of which is read by a pulse gated by the match bit. The read-out is to a set of buss lines connected to the Query Field Match Indicator Register position corresponding to the match-bit column. Each word of the read-only storage will have the corresponding row address permanently stored in it. Consequently, the result of a readout from one position will be the placing of that position's row address in the proper column of the Query Field Match Indicator Register.

Each column should have a detector for the condition of more than one match bit set in a column. Such a detector may be made from logical elements that sense the condition of at least two match bits in sequence. This may be found to be too slow for the matrix envisioned, and it is therefore felt that analog circuits might be developed for this job.

Query Field Control. The Query Field Control (QFC) determines the operations to be performed on each row of the matrix. Query Field Control can be broken down into two functions. First, boundaries within QDCS of the fields to be compared with the currently passing field must be defined. Second, the type of compare and linking to be performed must be determined.

The general operation is shown in Figure 11. The Field Record Control Counter (FRCC) is compared associatively with field numbers stored in the CAM Field Control (CFC). Successful comparison indicates that the field passing the cylinder scan heads is the field to be compared with the row of QDCS corresponding to the field number and sets the corresponding match bits in CFC. In turn,

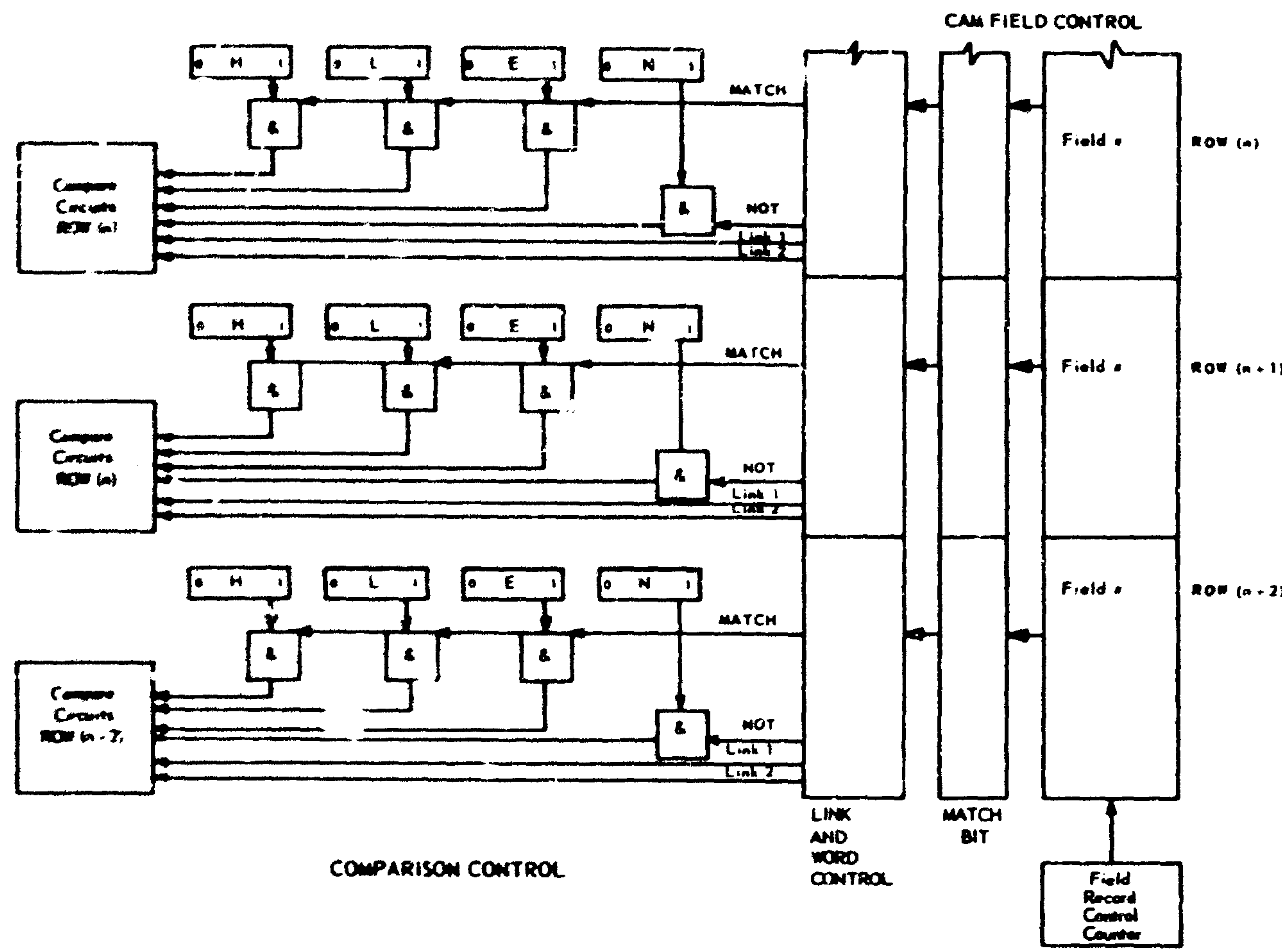


Figure 11. Query Field Control

the match bits operate through the Link and Word Control to bring up the "match" line and the Not, Link 1, Link 2 lines as required.

Comparison type is stored in the H (high), L (low), E (equal), and N (not) registers shown. The registers for storage of linking commands are located in the Link and Word Control. If the "match" line comes up and H, L, or E flip-flop is set, the corresponding function will be carried out in that row of the matrix. The Not function will operate if the N flip-flop is set. The Link operations are completely under control of registers in the Link and Word Control.

The Link and Word Control is shown in more detail in Figure 12. The function of this control is to ensure the scanning of only those words of QDCS which actually correspond to a particular word of the field currently passing the scanner heads. In the Figure, L1 and L2 are the registers in which the commands for Link 1 or Link 2 are stored when the control is first loaded for a scan operation. The End register must also be loaded to indicate the end of a between limits scan. The Field Number Match lines determine which field will be active at any one time. In turn, the step flip-flops determine which word of the field will be active at any time.

Operation of the circuit is best explained by example. Assume the field to be compared is one word long in QDCS row 2. The field number comparison sets the Field Number Match line of row 2. The Begin Field Reset line resets all step flip-flops to 1. Then the step flip-flop of row 2 in combination with the Not and Compare lines will set up the proper lines to the Comparison Control registers.

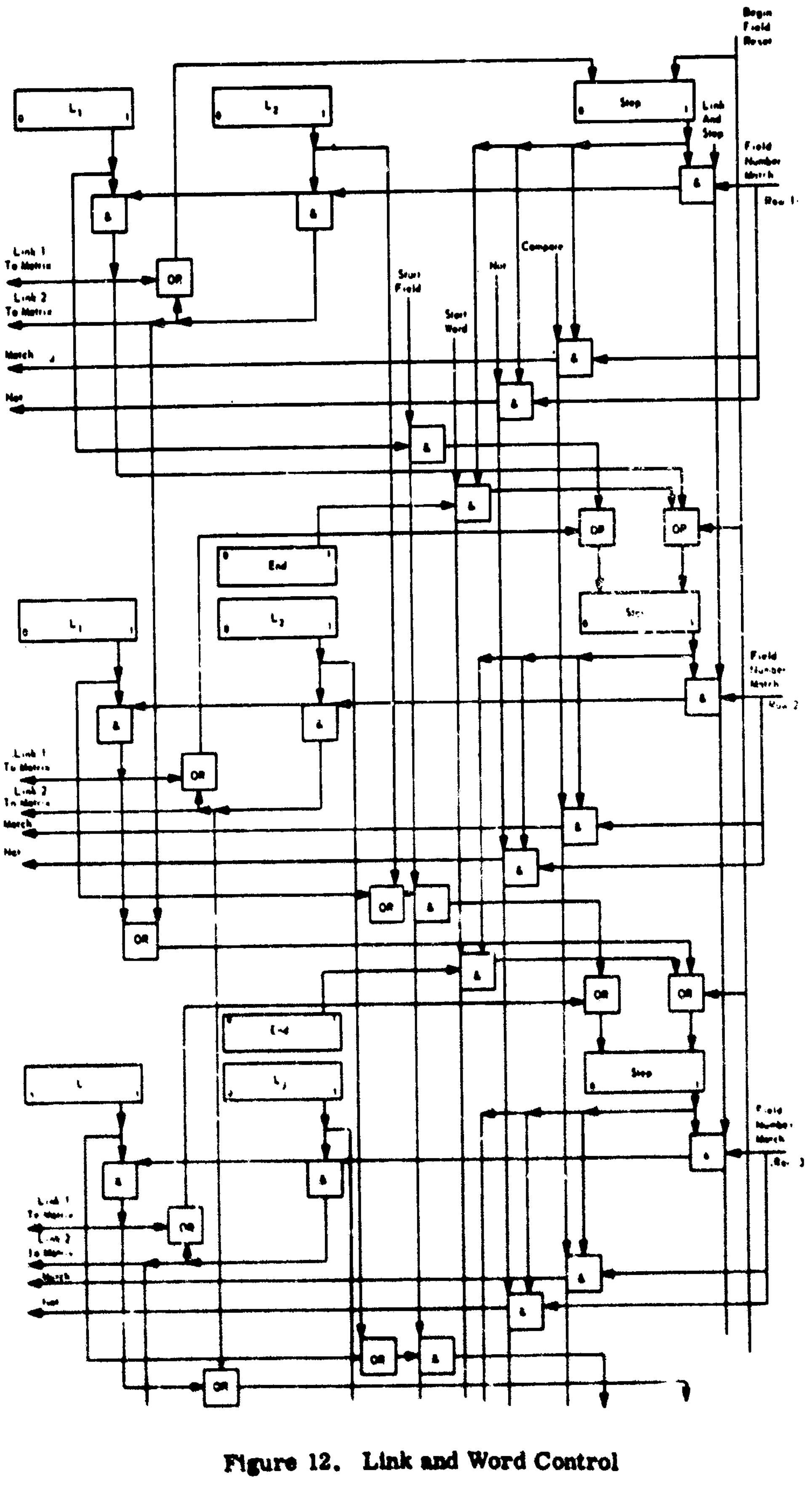


Figure 12. Link and Word Control

If the field to be compared is more than one word long, for instance a word in row 1 and 2, a linking operation will be required. Link 1 (L1) must be set in row 1. When Link 1 is set in word 1 the step flip-flop of row 2 will be turned off so that row 2 will be made inactive. At the end of the comparison of the first word the link step mechanism will turn off the step flip-flop of row 1 and turn on the step flip-flop of row 2. In addition the Link 1 line to the matrix will transfer the match/no-match condition of row 1 to row 2.

Link 2 operates in the same manner except that the linking is from one row to two below.

In a between-limits search the fields which bound the search are stored in alternating words. This simplifies the function so that it can be performed by two standard link operations and the END operation. The storage of the limit fields in QDCS is illustrated below:

Word 1: First word of upper-bound field
Word 2: First word of lower-bound field
Word 3: Second word of upper-bound field
Word 4: Second word of lower-bound field
etc.

The fields must therefore be linked by Link 2. In addition, the last word of the upper-bound must be linked by a Link 1 to the last word of the lower-bound field. The last word of the lower-bound field must also be marked by a 1 loaded into the END flip-flop. As can be seen in Figure 12, this END flip-flop permits a Link 1 which does not shut off the next lower row operation. This operation of the END flip-flop is only enabled on the last words of the field.

Query Field Match Indicator. The Query Field Match Indicator Register (QFMIR) is simply a flip-flop register large enough to hold one FCM row address for each column of the matrix, along with a mask bit for each column. For a FCM of 40 colums and 128 rows the QFMIR would comprise 320 bits at eight bits per column. The eighth bit (the mask) will only be set by the multiple match detector in the FCM. It will be used to ensure that fields which give multiple matches on a track will be assumed to satisfy all queries.

String Identity Section

Once individual fields have been found to compare with specified parameters, they must be associated with a particular query, and this function is performed by the String Identity Section (SIS). The major units of this section are the Record Control, the Identity Control, the Record Compare Matrix, the Query Identification Compare Store, and the Record Match Indicator.

The String Identity Section operation is very similar to that of the Field Compare Section just described. The differences in the two sections are in detailed operation of the various units. These differences in operation are necessary in order to allow for the treating of the strings of query identifications in QICS in the same manner as a single field is treated in the QDCS. In addition, changes in the compare circuits are required to properly treat inputs of multiple-match columns from the FCM.

Record Control. The Record Control in this section is modified by the String Identity Section Read-Out Control (SISROC) to allow inhibiting all string operations when no parameters are compared to a passing field. This control

senses the condition of the Query Field Control when a field is passing the heads. If no comparison is attempted on this field, the SISROC will prevent all operations of the Identity Control. This, of course, inhibits all operations in the RCM and the QICS. As soon as a field is passed in which at least one comparison is tried, the String Identity Section Read-Out Control will allow the whole SIS section to resume operation. In addition, since the actual comparisons made are only on one "character" per field, comparison control pulses will only be put out at the end of each field.

Identity Control. Figure 13 shows the general operation of the Identity Control (IC). This control has a link 1 and an Equal register for each row of the compare circuits of RCM. Both of these registers are loaded when the query strings are made up. The purpose of the Equal register is to enable or inhibit matching on its particular row. The Link 1, as in the Query Field Control, allows for operations on strings of more characters than are contained in a word of the Query Identity Compare Store.

In this section, control variations relating to particular fields selected for scanning have been made to depend upon an across-the-board decision. If any comparison was performed on the field in the Field Comparison Section, all comparisons will be enabled in the String Identity Section for this field. If no comparisons were performed in the Field Comparison Section, no comparisons will be allowed in the String Identity Section. It is apparent that the associatively addressed Field Number Selector present in the QFC is not required here. Instead, the Link and Word Control is driven directly by the String Identity Section Read-Out Control (SISROC).

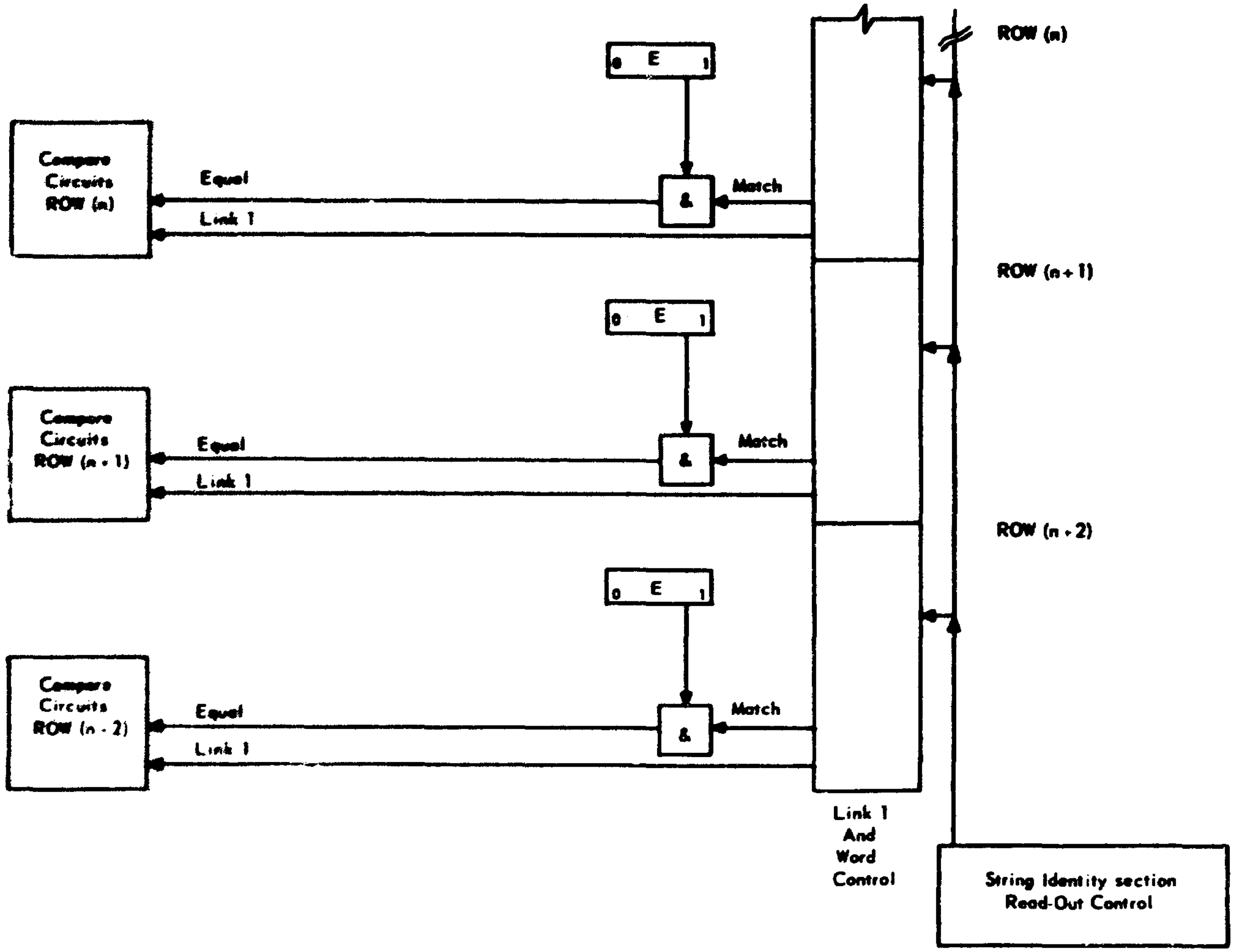


Figure 13. Identity Control

The operation of the Link 1 and Word Control is detailed in Figure 14. At the beginning of the record all the step flip-flops are set to one. Then if no Link 1 flip-flops are set, the level from the SISROC is gated by a Compare pulse and the row step flip-flop to the Match Line. If Link 1 is on, the Link 1 register modifies this operation in the succeeding row by turning off the row's Step flip-flop. Therefore, there will be no Match output from a row which has Link 1 set in the row above. Link and Step comes after each complete word has been read out of the QICS. In a linked chain, this will turn off the Step flip-flop set for the previous row compared and turn on the Step for the next row in line. Thus, in a linked chain defining a string, only one word of the string will be compared at a time.

Record Compare Matrix. The Record Compare Matrix (RCM) is simplified version of the Field Compare Matrix previously explained. The matrix is designed to compare selected rows of the Query Identity Compare Store with successive results from the Field Compare Matrix. At each row/column intersection, comparison and associated "read only storage" read-out circuits perform the required comparison. In addition, each intersection has a multiple-match indicator bit associated with it which will be used to indicate that multiple matches have occurred on a scanned field.

Operation of the comparison circuits is shown in Figure 15. The figure shows two match bits in a column. The inputs to one comparison circuit and the method of setting it are shown. The column inputs from the Query Match Indicator register are called field busses and are shown marked F, $\overline{F}$, and NOT MULTIPLE-MATCH FIELD BUSS. The row inputs from the Query Identity

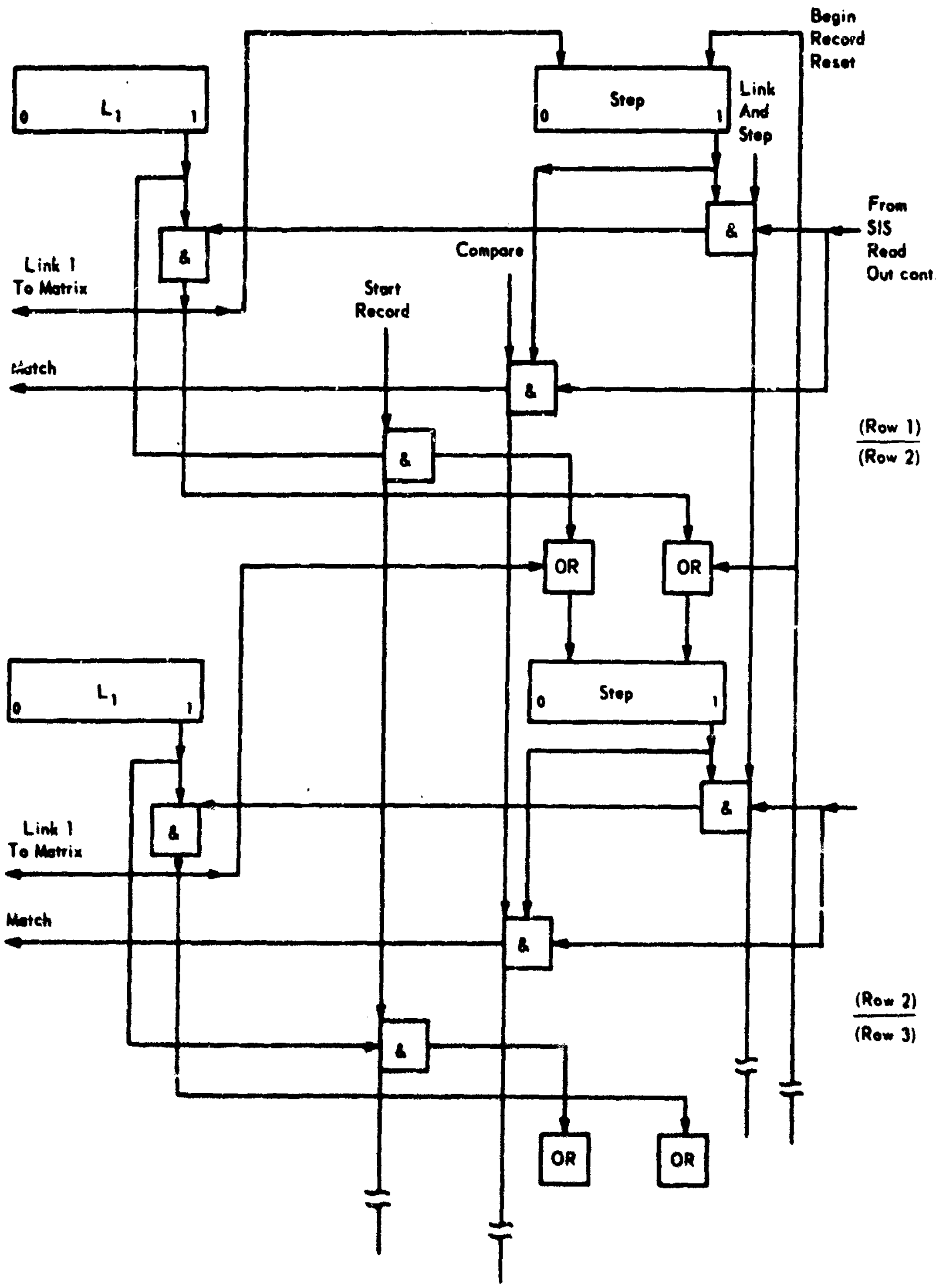


Figure 14. Link and Word Control for Identity Control

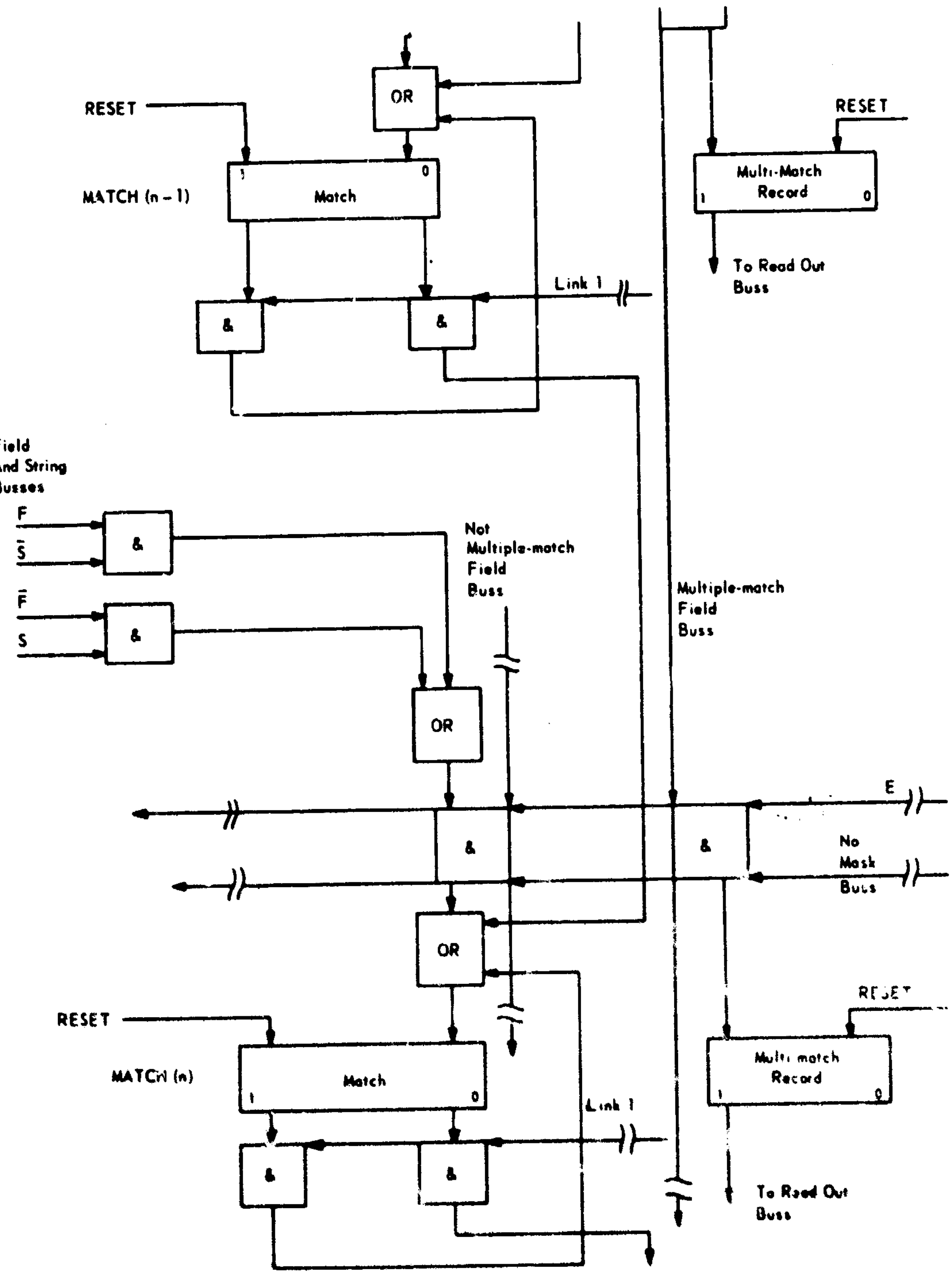


Figure 15. Record Compare Matrix Compare and Link Circuits

Compare Store are the string busses shown marked S, $\overline{S}$, and NO MASK BUSS. Here again, F stands for a 1 bit in the QM[T] while S stands for a 1 bit in the QICS.

Operation is as follows: At the start of the record, all match bits in rows containing Query Identity Strings are reset to a 1 or match condition. All other match bits in the matrix are set to ∅ or no match. The mask for the first QICS character is read into the mask register. Then the first bit from the first QICS "character" is read into the string busses. At the same time, the QFMIR reads out the condition of its Multiple Field Match indicator to the proper Multiple Field Match Buss. In any particular position, a no-match will occur if the Field and String Busses do not match, there is no masking, there is no multiple field match from QFMIR and the circuit is enabled by the EQUAL(E) line. If there is an indication of a multiple field match on a column, no mismatch will be recorded in the column, although the probability is high that the field and string busses are not equal. In this case, all those intersections enabled by E and No Mask will have their Multi-Match Indicator set.

Similarly, the rest of the word of QICS is compared with a new mask for every "character." Note that, contrary to the requirements for two masks per eight-bit character in the FCM, only one new mask is needed per "character" in the RCM. This is because the FCM is set up to handle both "packed" (four bits per digit) decimal and unpacked eight-bit BCD, while the RCM only handles binary (row address) "characters."

When the word is finished, if the string is longer than a word, the Link 1 operation transfers all No-Match conditions down one row. This next row is then enabled by E while the row just completed is set to No-Match and inhibited

by disabling its E line. Comparison then continues until the end of the record is reached.

When the end of record is reached, the contents of the Read-Only-Storage Registers associated with each row/column intersection in which a match bit is set are read into the Record Match Indicator (RMI). In addition, the Multiple Match Bits of any column which has at least one match bit set at the end of the record will be transferred to the RMI. At this point, the results in the Record Match Indicator define the records to be read out from this angular position and associate each record with the query that it answers.

Query Identification Compare Store. The Query Identification Compare Store operates very much like the QDCS previously described. The only difference is in the control of readout. As previously explained, the mask is only read out to the mask register once per "character." An additional feature is designed to save storage in QICS. Suppose that the identity strings of two queries were the following representing a query on a record of 11 fields:

Column in QRS / Name	1	2	3	4	5	6	7	8	9	10	11
A	2	5	M	M	7	M	8	M	M	M	M
B	1	3	M	M	M	M	9	M	M	M	M

M stands for masked out "characters." In such a case it would be a waste of space in QICS to store the full strings. Instead the strings should be stored as follows:

Columns in QICS / Name	1	2	3	4
A	2	5	7	8
B	1	3	M	9

leaving out the completely masked characters.

Such storage of the strings is made possible by the String Identity Section Read-Out Control (SISROC). As previously stated, this control inhibits all string operations when no parameters are compared to a passing field. Thus, irrelevant fields will have passed by without being tested. In the second example strings shown, the String Identity Section would pause between columns 2 and 3 and from column 4 to the end of the record.

Record Match Indicator. The Record Match Indicator is identical in form and function to the Query Field Match Indicator and needs no further description here.

Data Transfer

Once the scan has found records to read out the actual data must be transferred to the central processor memory. Recall that the read/write heads are positioned some distance around the cylinder from the scanner heads. They

follow the scanner heads by at least the maximum number of characters that a physical record is ever expected to have (assumed 1000 in this case). To accomplish readout under control of the scanner, a record must be kept of the channels to be read beginning with each angular position at which a set of records starts. This record keeping is done in the APS/STATUS/TAG Store.

This Store is a conventional word-organized memory. If organized as shown on the basis of two words per entry, its maximum size would be:

$$2\text{x} \; \frac{\text{maximum character capacity of a track}}{\text{number of characters in smallest record allowed}}$$

When the scanner finds a position where at least one record required is stored, it makes an entry in the APS/STATUS/TAG store as follows: The angular position where reading should start is kept in the Angular Position Store (APS). The channels to be read starting at that position are marked by bits in the Status Store (SS) which has one bit within each register for every channel. For each APS register there is also a corresponding register of tags in the Tag Store which are used to record the contents of the RMI. In Figure 8, five tags per APS were assumed as an illustration. In this tag register, the tags are stored corresponding to up to five tracks to be read from the angular position stored in the APS of the entry, and it is also assumed that five data channels are available to the main processor memory.

Normally, the number of records marked at a single APS will not exceed five. In this case, a single revolution of the cylinder will suffice to read all relevant records into memory.

Main memory may be too full to take all records found in a single revolution or there may be too many records found (more than five) at one angular position, In this case, readout may be conducted by the computer as follows: On the first revolution of the cylinder, the APS and its complete Status Register and a maximum of five Tag positions per APS will be stored in an entry. Tags stored will always be those corresponding to the left-most group of set status bits. Any or all of the five channels with tags may be read out on request of the computer. When the corresponding track is read out, the status bits will be reset to Ø and cannot be set again until no more status bits remain set. On a second revolution, tags generated by a second scan will be read into the tag register for up to five of the leftmost status bits remaining set. Again, reset of status bits will occur on read out. For example, suppose that at angular position 300 a group of seven records were to be read out. These records have tags here designated as A, C, F, I, J, L and P. They are located on tracks 1, 2, 5, 10, 15, 20 respectively. The complete entry in the store after the first scan would be:

APS: 300

SS: has 1's in bit positions 1, 2, 5, 7, 10, 15 and 20 and Ø's everywhere else

TAGS: A, C, F, I, J.

The first five records would be read out and the first five status bits reset. After a second scan the entry in the store would be:

APS: 300

SS: has 1's in bit positions 15 and 20 and Ø's everywhere else

TAGS: L, P

These last two records would be read out on this revolution.

A record is kept to indicate when there are more records to be read, and scanning of the cylinder will continue until the last record is read. At this time, scanning can be initiated on the next cylinder.

Writing is done similarly. Empty spaces are found through one revolution. As they are found, tags are inserted in the Tag Store. When the write heads come to the proper position, the record is written, the Status is reset, and the computer is informed of the action.

Each tag will be made up of the row address from the RCM along with a tag number (1 to 5). Under normal circumstances, the tag number will be irrelevant. But, when data is transferred which cannot be associated with one string (multiple matches) the tag number will be used to store the data in a particular place in memory. Thus, if two records from two tracks should be indeterminate simultaneously they will retain unique identifications.

Compare Controls

A summary of compare controls is given below. The normal read/write control instructions are not given. The underlined letters correspond to the flip-flops in which the commands are stored as shown in Figures 11 and 12.

Equal Compare: Exact match between track and QDCS fields on all characters compared in this row sets the Match bit at the intersection of this row and the track(s).

High Compare: If track data field characters compared are greater than corresponding characters in QDCS, the Match bit will be set at the intersection of this row and the track(s).

Low Compare: If track data field characters compared are less than corresponding characters in QDCS, the Match bit will be set at the intersection of this row and the track(s).

Link 1: Sets the Match bit of the next lower row to No Match if Match bit of this row is No Match, otherwise does not change condition of next Match bit. Prevents matching on all succeeding rows up to first row with no Link 1 preceding it. Therefore, last word of a field must not have link 1 set. As words of a field are passed, the succeeding Link 1 program gates are successively opened, while preceding Link 1 gates are closed. This allows one word at a time of a field to be compared.

Link 2: Sets the match bit of the second following row to No Match if Match bit of this row is No Match, otherwise does not change condition of next Match bit. Prevents matching on all words of a field except that being presently passed.

Not: Complements the Match bits of the row which have been set up by High, Low or Equal compare. This gives the functions Not High, Not Low, or Not Equal, respectively. "Not" is meaningless without one and only one of the comparison instructions.

END: The end of the field of the lower stored parameter of boundary parameter fields in a between-limits scan is indicated by the END command.

EXAMPLE PROBLEM

Specifying an example file, the following discussion compares the operation times required to process a given number of individual transactions on both a conventional machine with standard disk storage and the file processor just described. Comparisons are also made using two slightly different file organizations to point out some of the advantages of the file processor.

In its upper and lower parts, Table XV describes the access models for both the associative and conventional machines reppectively. The models typify a number of operations which will be performed in the general process of reading or writing data, and the times associated with each operation. Each operation is identified by a number or symbol. These symbols will be used later in the Operation Tables to explain and time the query update functions. In Table XV, the operations described are almost self-explanatory. The term Random Access implies that the access arm skips over some random number of cylinders before it settles down to actually read data. The term Sequential Access implies that the arm moves from one cylinder to the next in succession. The term Record Space means just that: the associative machine is able to insert data in a given space without having to rewrite any of the rest of the track.

The times given are in terms of "access times." One access time is defined as the time required for one revolution of the disk in question, and this time is assumed the same for both disk systems considered. In the Associative section of Table XV, the times are somewhat lower than the sum of operation times to do comparable functions in the Conventional Machine section.

The reason for this is that the associative machine is usually looking for several records at once, while the conventional machine is always looking for some specific record. The time given for sequential readout of records in the associative machine is two accesses per cylinder read, since in these problems no more than one revolution will be needed to read out all records selected from a cylinder.

What might be considered a typical Activities File is defined by Table XVI as the Commercial Vehicle Activities File. This file will be used in both examples presented. It is a file of 270,000 logical records, each of which is assumed to have one fixed and, on the average, five periodic groups. These groups all exist as separate physical records; all are chained and indexed as explained later in each example.

In the two examples which follow, 10,000 transactions will be processed against this file. These transactions are described in Table XVII where the update transactions are described for both examples in the upper section, and the queries, which differ for examples 1 and 2, are described in the lower sections. In the table, the percentages given refer to the mix of types among updates alone and among types of queries alone. The intermix of percent query and update will be given in the individual problem results.

The record format of the file for the first example is presented in Figure 16. The file is organized in two sections: the fixed groups of all records are stored as one set of cylinders, the periodics on another. In a given logical record, the logical connection between fixed and periodic groups and between periodics of the same logical records is an address chain which includes the ICC number and

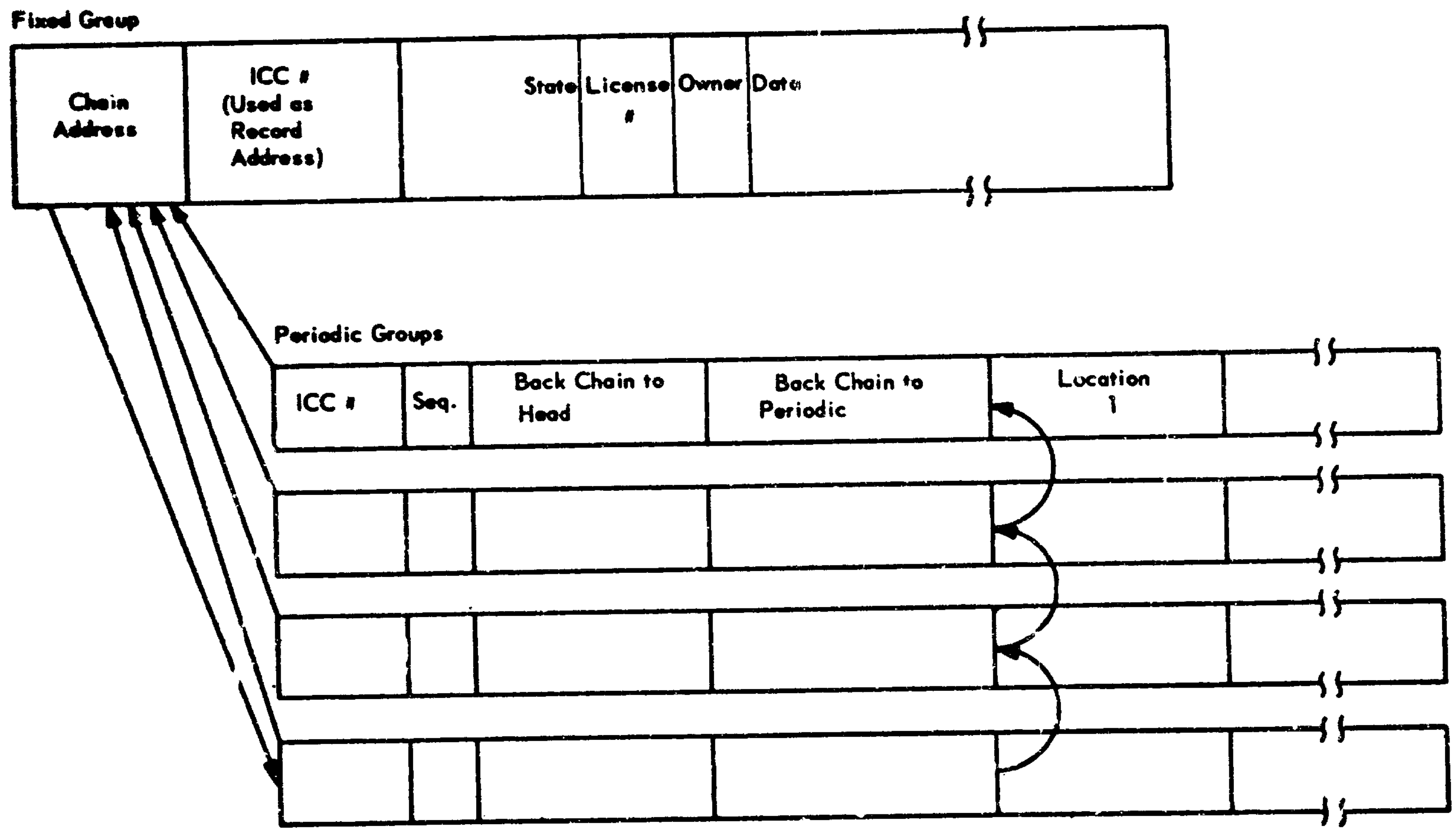


Figure 16. Record Organization for Both Machines
Example 1

other identification characters (explained below). The chain address in the fixed group refers to the last stored periodic group of the logical record. Each periodic is chained to the periodic immediately preceding it in the storing chronology of the logical record. In addition, each periodic is chained to the header of the record.

The fixed groups are in order by "ICC number", the periodics by "location". Only two indexes are kept, though in the two machines the indexes are to different levels. The indexes are to the ICC numbers of the fixed group and to the locations of the periodics.

If the ICC number is given, the general procedure for reading a complete record will be to find the proper fixed group by the ICC number index. This fixed group will be accessed, and the forward chain address will then be used to access and read the last-stored periodic group. Using the back chaining of periodic groups, the machine can then read the next periodic in the chain. The process continues until the end of the chain is reached.

The specific record parameters for the organization used by the associative processor in both examples are shown in Table XVIII. The addressing parameters shown in the table are given in cumulative form. For instance, the chain address in the fixed group referring to the corresponding periodic group chain will comprise the ICC number, two characters of cylinder information, and the sequence character—nine characters in all.

The specific record parameters for the file organization used by the conventional machine in Example 1 are shown in Table XIX. This table is self-explanatory. The differences in addressing, which seem comparatively minor,

are required by the greater degree of address definition needed in the conventional machine. It should be noted that this finer definition is a definite disadvantage in the update procedure, since it will require more update in operations for maintenance.

Observation of the index portion of Tables XVIII and XIX will bring out the following differences. In the associative machine, the indexing is only to the cylinder level. In the conventional machine, indexing is to the track level by a combination of indexes. The cylinder index allows quick access to the cylinder on which the record is located; then a track index stored on the same cylinder as the record allows quick access to the correct track. This combination of indexes is used to give quick access to a track during retrieval, but at the same time to minimize random access arm motion which might otherwise be required during update of a track index stored on a special cylinder.

In the problem examples, two assumptions were made. No cylinder index updating was required. On the other hand, record indexes were always updated when changes to records were made.

In Table XVII, the query and update functions to be performed in the first example were specified. In Table XX, the actual operations required of the associative machine in the performance of any one function is specified. To understand the table symbols, reference should be made to the access model for the associative machine specified in Table XV. The operation numbers specified in Table XV are used to define the operations required to perform a

function as specified in Table XX. For instance, Table XX shows that the function ADD a RECORD requires the following operations:

Operation Number	Operation
3.	Random Access and Read Complete
1.	Random Access and Write Into Empty Record Space
1.	Repeat 1 Above

for a total time for the function of 16.45 accesses. When a particular operation is to be repeated n times in a table entry, the symbol for the operation is followed by (n). An example of this occurs in Query Type 5 where 8(123) implies that operation 8 is repeated 123 times. Repetition of a set of operations is denoted by enclosing the set in parenthesis followed by the receptition number in parenthesis.

Table XXI specified the conventional machine operations to be performed for each query and update function of Example 1. The table in conjunction with Table XV is self-explanatory.

The results of Example 1 are given in Tables XXII and XXIII. In the upper part of these tables the total times for each type of transaction are given per 10,000 transactions. For instance, of 10,000 total updates of which 1666 transactions or 16.66% were Type 1, the total time taken for those updates would be 27×10^3 access times. In the lower part of the tables the 10,000 transactions are broken down to give total transaction time as a function of the percent of the total which constitutes update time. In addition, Table XXII gives a ratio of the comparative performance of the two machines. The performance ratio is defined to be

$$\frac{\text{Conventional Machine Access Time}}{\text{Associative Machine Access Time}}.$$

In Figure 17 the same results are shown in graphical form. Note that the left-hand scale for associative machine times is scaled down from the right-hand scale for the conventional machine by a factor of 5. The ratio scale is in the center.

A study of these results points to the great advantage of the associative processor when any appreciable scanning is required in queries. For this reason Example 2 was prepared. Example 2 is quite similar to Example 1 except that the file organization, as shown in Figure 18, for the conventional machine has been changed to allow for cross-chaining by "Owner" in the fixed groups and by "Route Entered" in the periodics. Cross-chaining creates chains by subjects which span several of the normal file logical records. Indexes to the cross-chain entry points are also provided. To get to an entry by cross-chain, therefore, the cross-chain index must be accessed to find the first record in the chain. Then the first chain record is accessed and scanned. From this point, successive records in the chain are accessed until the proper record is found. In this example no particular order is maintained in the cross-chain.

In Table XXIV the parameters for the conventional machine file organizations are shown. Table XXV shows the conventional machine operations required in the Example 2 functions, and Table XXVI shows the conventional machine results on 10,000 transactions. In addition, the results of comparison to the associative machine results of Table XXIII are shown in ratio form. The results are also shown graphically in Figure 19.

Examination of the results clearly shows the advantage of the file processor as a function of the amount of scanning required on queries. In the two examples,

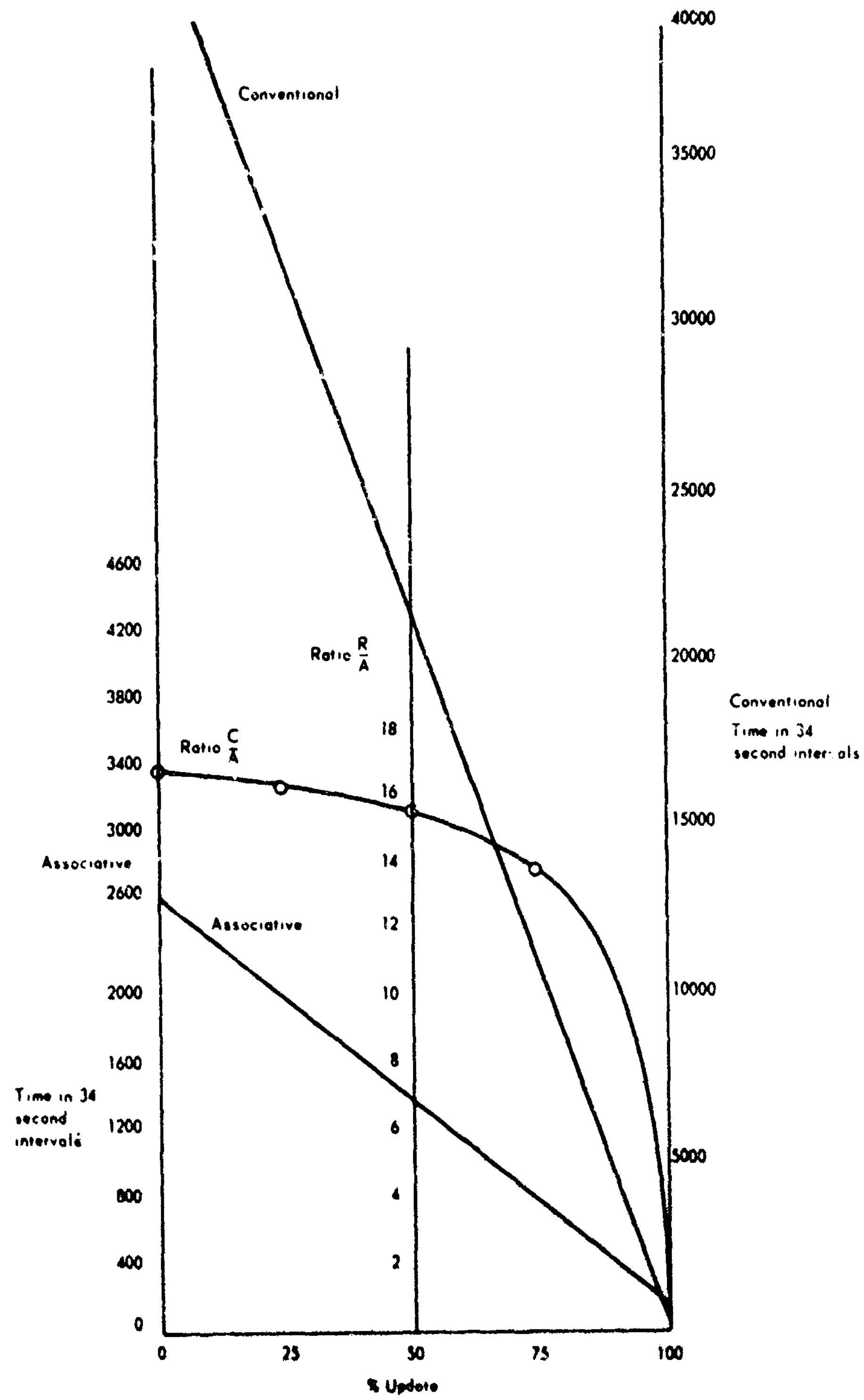


Figure 17. Results of 10K Transactions Against a File of 270K Logical Records

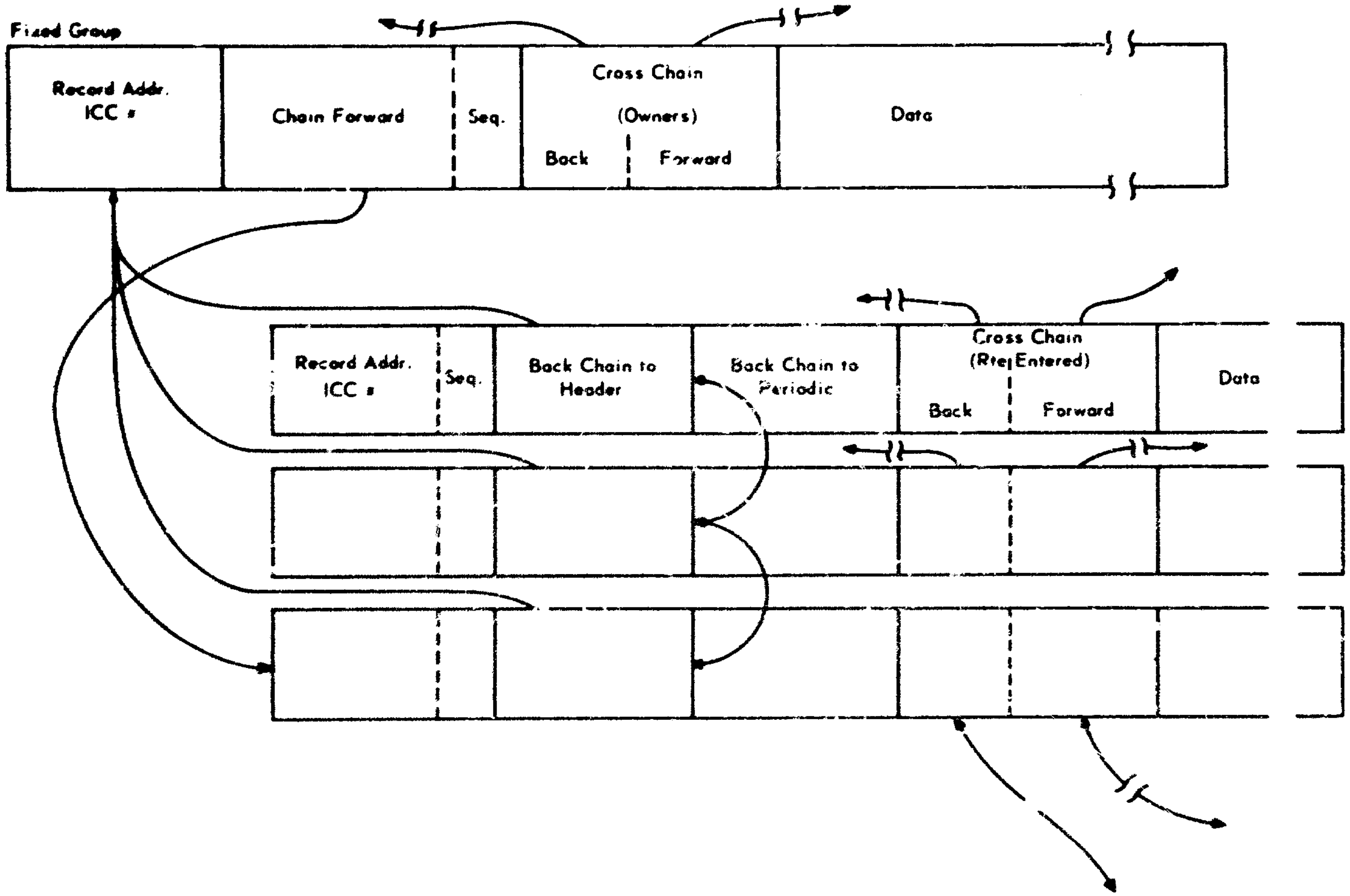


Figure 18. Record Organization for Conventional Machine
Example 2

consider the speed advantage of the associative machine in the case of queries only. The advantage gain of the associative machine from 2.2 in Example 2 to 16.8 in Example 1 is almost wholly a result of an increase in scanning required in 20% of the queries.

It is further evident that the queries considered were weighted in favor of queries mentioning the ICC number and locations, the keys which were indexed and about which the file was organized. That is, 80% of the queries mentioned an indexed field. Since this percentage of simple queries appears to be a somewhat high, it can be argued that the figures of merit in the second example should be closer to 3- or-4 to 1 in favor of the associative machine.

Another point which should be pointed out is that the transactions in the examples were processed on a one-at-a-time basis. The associative machine's multiple input capability was not used to great advantage. As a matter of fact, the processing as stated could have been done in a scanner of at most 16 six-character words in QDCS and 2 four-character words in QICS. It is felt that more realistic use of the scanner would considerably increase the speed gain of the associative machine.

DISCUSSION AND RECOMMENDATIONS

A storage and retrieval system for formatted files in intelligence applications must meet stringent requirements for capacity, speed, and flexibility. Consider how the file storage device described meets these requirements.

The device has a high storage capacity. Fitted on the IBM 1302 Disk Storage Unit, for instance, the system could store 180 million characters. Or,

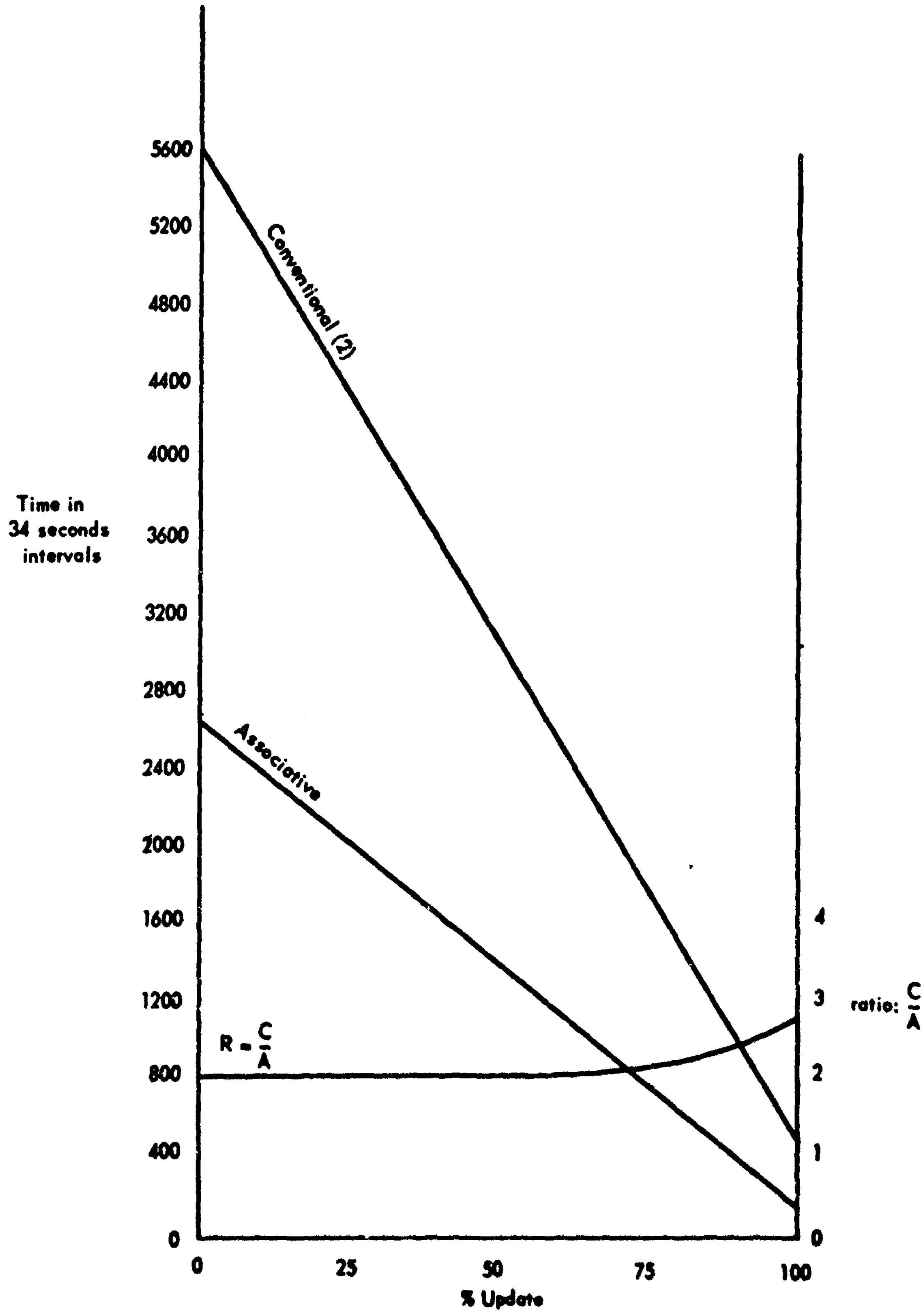


Figure 19. Results of 10K Transactions Against a 270K Logical File

if the system were fitted on a disk system with removable disks the storage areas is, for all intents and purposes, limitless. Another way in which this device could be used is as a buffer for other devices which could be read and written in parallel. As such, the capacity is limited only by the storage medium buffered.

Storage and retrieval speeds of the scanner are high. They depend primarily upon the speed of the storage medium past the heads and upon the number of tracks which can be read at once. Even with the very small scanner assumed in the examples, speeds of storage and retrieval were increased over those of present-day disk storage systems by factors of from 2 to 16, and possibilities for even greater speed exist in different organization and use of storage.

The processor has great flexibility both in the data it can handle and in the operations it can perform. The data is somewhat restricted in that it must be stored in parallel fields on fixed-length physical records. But such records can be logically joined to form quite complicated variable-length logical records in a several ways. The linking features allow fields of any length up to the maximum size record or to the maximum number of characters in QDC.

The device is also flexible in its logical abilities. The logical connectives AND, OR, and NOT, along with HIGH, LOW, EQUAL and BETWEEN LIMITS compares, furnish capabilities to handle complicated logical searches expressed in disjunctive normal form. If the query contains parameters which cannot be so expressed, thus resulting in multiple matches on a track record, all records which cannot be resolved by the simple scanner can be read into memory where they can be further analyzed.

The processor is well able to handle inputs on an individual basis or in batched form. The amount of batching possible depends upon the number of rows in the Field Compare and String Identity Sections.

The most important feature of the design is its independence from the CPU while scanning. This will save a good deal of CPU time, which can be used for more complicated types of processing than simple file scanning.

The unit is comparatively simple in concept and appears simple to implement. It could be built in modular form to handle varying storage requirements. No obvious circuit timing limitations have been found, and the timing of the comparisons and string identification do not change appreciably as more equipment—such as more disks and scanner heads—is added. Electronically, the timing for the complete compare matrix is essentially the same as for one compare circuit. Should the system require expansion, no timing problems due to cascaded circuit delays would become evident. Equipment has already been built to read/write in parallel on 40 disk tracks at a time, and there should be little problem in building at least a small scale device to operate on a present-day large-scale computer.

In conclusion, the design presented seems well suited to the storage and retrieval of formatted files on an economical medium. Its added cost seems little enough to pay for the large gain in performance over conventional disk storage. It is therefore recommended that a small system of this sort be further defined and, if it still seems feasible, built in prototype form.

Table XV. ACCESS MODELS

Associative Machine

Operation	Operation	Average Time in Accesses
1	Random Access Write into Empty Record Space	5.35 a
2	Random Access Insert in full Record Space	6.75 a
3	Random Access Read Complete	5.75 a
4	Random Access Read Chain Addv	5.50 a
5	Sequential Access Write into Empty Record Space	2.35 a
6	Sequential Access Replace full Record Space	3.75 a
7	Sequential Access Read Complete	2.75 a
8	Sequential Read Out of Records	2.0/cyl.

Conventional Machine

Operation Symbol	Operation	Average Time in Accesses
R	Random Access Motion	5.0
SSR	Successive Reads on Same Cylinder Reading by Record	0.5
SSRI	Successive Reads on Same Cylinder Reading Complete Track	1.0
SSW	Successive Write on Same Cylinder Record Mode	0.5
SSRT	Read Records from Same Track	1.0/track read
S	Access Next Successive Cylinder	2.0

Table XVI. COMMERCIAL VEHICLE ACTIVITIES FILE

Field Name	Fixed Fields Number Of Char. Packed	Type	Number Of Different Values	Meaning
License	7	Alpha	25000	License number
State	5	Alpha	50	Abbreviation of License state
ICC number	5-1/2	Num	270K	ICC registration number
Company	18	Alpha	5000	Owner's name
Company number	7	Alpha	1000	Company vehicle number
Type	9	Alpha	100	Type - Truck, taxi, etc.
Make	9	Alpha	40	Make (Chevy, Ford, etc.)
Empty Wt.	2-1/2	Num	180	Expressed in pounds
Passenger Cap.	1-1/2	Num	200	Number paying passengers possible
Freight Cap.	3	Num	1000	Expressed in pounds
	67			

Periodic Fields

Time	10	Y/M/D	365	Time
Observer	5	Alpha-Num	1000	Observe code
Location I	18	Alpha-Num	5000	Road & state observers station
Location 2 (St. No.)	2-1/2	Num	20	Number
Type Rep	3	Alpha	10	Radio, Message, Talk to Driver

Table XVI. COMMERCIAL VEHICLE ACTIVITIES FILE (Cont'd)

Field Name	Fixed Fields Number Of Char. Packed	Type	Number Of Different Values	Meaning
Direction	3	Alpha	16	Compass points
Speed	2-1/2	Num	90	Miles per hour
Reason	3	Alpha	10	Codes for stops
Route (from)	18	Alpha-Num	500	Name or route left
Route (to)	18	Alpha-Num	500	Name or number of route entered
Freight Inv	3	Num	1000	Express in pounds
Passenger Inv	1-1/2	Num	200	Actual cost
Freight	1-1/2	Num	4	Quarters full
Passenger Est	1-1/2	Num	4	Quarters full
Type Cargo	3	Alpha	1000	Code for general type
	91			

Table XVII. TRANSACTION MODEL

Updates: Examples 1 and 2

Type	Action	Percent
1.	Add one record	16.6
2.	Delete one record	16.6
3.	Add one periodic	66.8

Queries: Example 1

Type	Query	
1.	Given ICC range, print fixed groups	5
2.	Given ICC number, print fixed groups	5
3.	Given ICC number, print last periodic group	10
4.	Given ICC number, print itinerary	10
5.	Search entire fixed portion of file	10
6.	Given two locations, print "from - to" groups	10
7.	Given location, print all fixed groups	20
8.	Given location, print all periodics	20
9.	Search entire periodic portion of file	20

Queries: Example 2

1-4	Same as Example 1	30
5.	Search on Owner Chain, find 325 records	10
6-8	Same as Example 1	50
9.	Search on Route Chain, find 500 records	10

Table XVIII. ASSOCIATIVE MACHINE FILE ORGANIZATION

Example 1

Type	Field	Length
Fixed Group:	Record Address: ICC number	6 char
	Forward Chain Address: ICC number + Cylinder +	2 char
	Current Last-In-Sequency Indicator	1 char
	Data	61 char
	Total	70
Periodic Group:	Record Address: ICC number + Sequence Indicator	7 char
	Back Chain to Header: ICC number + Cylinder + Track	2 char
	Back Chain to Periodic: ICC number + Sequence Indicator + Cylinder	3 char
	Data	85 char
	Total	97
Average Logical Record:	Fixed Group + Five Periodic Groups	565 char
Indexes:	Fixed Cylinder Index: to ICC number	1 Track
	Periodic Cylinder Index to Lowest Location on Cylinder	6 Track
Totals:	Fixed Data and Track Index	120 Cylinders
	Periodic Data and Track Index	835 Cylinders

Table XIX. CONVENTIONAL MACHINE FILE ORGANIZATION

Example 1

Type	Field	Length
Fixed Group:	Record Address: ICC number	6 char
	Forward Chain Address: ICC number + Cylinder + Track + Sequence	4 char
	Current Last-In-Sequence Indicator	1 char
	Data	61 char
	Total	72 char
Periodic Group:	Record Address: ICC number + Sequence Indicator	7 char
	Back Chain to Header: ICC number + Cylinder + Track	3 char
	Back Chain go Periodic: ICC number + Sequence Indicator + Cylinder + Track	4 char
	Data	91 char
	Total	105 char
Average Logical Record:	Fixed Group + Five Periodic Groups	597 char
Indexes:	Fixed Cylinder Index: to ICC number	1 Track
	Fixed Track Index to ICC number: 1 Track on Each Cylinder of Fixed Groups	125 Tracks
	Periodic Cylinder Index to Lowest Location on Cylinder	6 Tracks
	Periodic Track Index to Largest on Each Track or to Overflow Track	910 Tracks

Table XIX. CONVENTIONAL MACHINE FILE ORGANIZATION (Cont'd)

Example 1

Type	Field	Length
Totals:	Fixed Data and Track Index	125 Cylinders
	Periodic Data and Track Index	910 Cylinders

Table XX. ASSOCIATIVE MACHINE OPERATIONS

Examples 1 and 2

Function	Operations From Access Model	Times in Accesses
Add Record:	3, 1, 1,	16.45
Add Periodic:	3, 2, 1,	17.85
Delete Record:	3, 1, 1, 1, 1	27.15
Query Type 1:	3, 3, 8	13.50
Query Type 2:	3, 3	11.50
Query Type 3:	3, 4, 3	17.00
Query Type 4:	3, 3, 3, 3, 3	28.75
Query Type 5:	3, 8 (123)	251.75
Query Type 6:	3, 3, 9, 3, 8	21.25
Query Type 7:	3, 3, 8	13.5
Query Type 9:	3, 8 (860)	1725.75

Table XXI. CONVENTIONAL MACHINE OPERATIONS

Example 1

Function	Operations from Access Model	Times in Accesses
Add Record:	R, SSRi, R, SSR, R, SSR, SSW, R, SSW	25.0
Add Periodic Group:	R, SSRi, SSRi, R, SSR, SSR, R, SSR, SSW, R, SSW	25.5
Delete a Record:	R, SSRi, R, SSRi, SSR, SSW, R, (R, SSR, SSW, SSR SSW) (2), (R, SSR) (2)	36.0
Query Type 1:	R, SSRi, R, SSRi,SSRT	18.5
Query Type 2:	R, SSRi, R, SSRi, SSR	12.5
Query Type 3:	R, SSRi, R, SSRi, SSR, R, SSR	18.0
Query Type 4:	R, SSRi, R, SSRi, SSR, (R, SSR, SSR) (2), (R, SSR) (2)	
Query Type 5:	R, SSRi, R, (SSR(40), S) (125)	5135.5
Query Type 6:	R, SSRi, R, SSRt (13), R, SSRt(13)	42.0
Query Type 7:	R, SSRi, R, SSR, SSR, R, SSRi, (S, SSRi) (119)	451.5
Query Type 8:	R, SSRi, R, SSRi, SSRt (300)	24.0
Query Type 9:	R, SSR, R, (SSE(40), S) (910)	37321.0

Table XXII. CONVENTIONAL MACHINE TRANSACTION TIMES IN ACCESSES

Example 1

Update Time			Query Time			
Number of Transactions	Unit Time in Accesses	Time in Accesses	Type	Number of Transactions	Unit Time In Accesses	Time in Accesses
1666	25.0	41.6×10^3	1	500	18.5	9.25×10^3
1666	36.0	60.0×10^3	2	500	12.5	6.25×10^3
6668	25.5	170.5×10^3	3	1000	18.0	18.0×10^3
10000		272.1×10^3	4	1000	30.5	30.5×10^3
			5	1000	5135.5	5135.5×10^3
			6	1000	42.5	42.5×10^3
			7	2000	451.5	903.0×10^3
			8	2000	24.0	48.0×10^3
			9	1000	37321.0	37321.0×10^3
				10000		13514.0×10^3

10,000 Transactions Breakdown by %

Update		Query			
Percent	Time in Accesses	Percent	Time in Accesses	Total Times	Ratio of Performance
100	272.1×10^3	0	0	273×10^3	1.4
75	204.0×10^3	25	10878×10^3	11082×10^3	14
50	136.0×10^3	50	21757×10^3	21893×10^3	15.7
25	68.0×10^3	75	32635×10^3	32703×10^3	16.4
0	0	100	43514×10^3	43514×10^3	16.8

Table XXIII. ASSOCIATIVE MACHINE TRANSACTION TIMES IN ACCESSES

Examples 1 and 2

Update Time

Type Update	Number of Transactions	Unit Time In Accesses	Total Time in Accesses
1.	1666	16.45 =	27.0×10^3
2.	1666	27.15 =	45.2×10^3
3.	6668	17.85 =	119.0×10^3
	10000		191.2×10^3

Query

Type	Number of Transactions	Unit Time In Accesses	Time Time in Accesses
1	500	13.5	6.7×10^3
2	500	11.5	5.75×10^3
3	1000	17.0	17.0×10^3
4	1000	28.75	28.75×10^3
5	1000	251.75	245.75×10^3
6	1000	21.25	21.25×10^3
7	2000	259.5	591.0×10^3
8	2000	13.5	27.0×10^3
9	1000	1725.7	1725.75×10^3
	10000		2579×10^3

10,000 Transactions Breakdown by %

Update Time

Percent	Time in Accesses
100	191.2×10^3
75	143.4×10^3
50	95.6×10^3
25	47.8×10^3
25	0

Query Time

Percent	Time in Accesses	Total
0	649×10^3	191×10^3
25	649×10^3	792×10^3
50	1299×10^3	1395×10^3
75	1948×10^3	1996×10^3
100	2597×10^3	2597×10^3

Table XXIV. CONVENTIONAL MACHINE FILE ORGANIZATION

Example 2

Type	Field	Length
Fixed Group:	Record Address: ICC number	6 char
	Forward Chain Address: ICC number + Cylinder + Track + Sequence	4 char
	Current Last-In-Sequence Indicator	1 char
	Cross Chain	20 char
	Data	61 char
	Total	92 char
	Average Cross Chain Length	54 records
Periodic Group:	Rceord Address: ICC number + Sequence Indicator	7 char
	Back Chain to Header: ICC number + Cylinder + Track	3 char
	Back Chain to Periodic: ICC number + Sequence Indicator + Cylinder + Track	4 char
	Cross Chain Route Entered	20 char
	Data	91 char
	Total	125 char
	Average Cross Chain Length	540 records
Average Logical Record:	Fixed Group + Five Periodic Groups	717 char
Indexes:		

Table XXIV. CONVENTIONAL MACHINE FILE ORGANIZATION (Cont'd)

Example 2

Type	Field	Length
Indexes:	Fixed Cylinder Index: to ICC number	1 Track
	Fixed Track Index to ICC number: 1 Track on Each Cylinder of Fixed Groups	159 Tracks
	Fixed Owner Index to Record Level (last entry)	13 Tracks
	Periodic Cylinder Indextto Lowest Location on Cylinder	6 Tracks
	Periodic Track Index to Largest on Each Track or to Overflow Track	1078 Tracks
	Last Route Index to Record Level	4 Tracks
Totals:	Fixed Data and Track Index Periodic Data and Track Index	159 Cylinders
	Periodic Data and Track Index	1078 Cylinders

TABLE XXV. CONVENTIONAL MACHINE OPERATIONS

Example 2

Function	Operations from Access Model	Times in Accesses
Add Record:	R, SSRi, SSRi, SSRi, SSRi, R, SSRi, R, SSR, SSW, R, SSR, SSW, R, SSR, SSW, R, SSW, R, SSW, SSW	51.5
Add a Periodic:	R, SSRi, SSRi, SSRi, R, SSRi, SSR, R, SSRi, SSW, R, SSR, SSR, R, SSR, R, SSR, R, SSR, SSR,	46.0
Delete a Record:	R, SSRi, R, SSRi, SSR, SSW, R, SSR, SSW, R, SSR, SSW, (R, SSR, SSW, SSR, SSW) (2) R, SSR, SSW, (R, SSR, SSW)(9)	96.0
Query Type 1:	Same as in example 1	18.5
Query Type 2:	Same as in example 1	12.5
Query Type 3:	Same as in example 1	18
Query Type 4:	Same as in example 1	30.5
Query Type 5:	R, SSRt(3), (R, SSR) (325)	1793.5
Query Type 6:	Same as in example 1	42.5
Query Type 7:	Same as in example 1	451.5
Query Type 8:	Same as in example 1	24.0
Query Type 9:	R, SSR, (R, SSR) (500)	2756.5

TABLE XXVI. CONVENTIONAL MACHINE TRANSACTION TIMES IN ACCESSES

Example 2

Update Time			Query Time			
Number of Transactions	Unit Time In Accesses	Time in Accesses	Type	Number of Transactions	Unit Time In Accesses	Time in Accesses
1666	51.5	85.5×10^3	1	500	18.5	9.25×10^3
1666	96.0	160.0×10^3	2	500	12.5	6.25×10^3
6668	46.0	308.0×10^3	3	1000	18.0	$18.- \times 10^3$
1000		554×10^3	4	1000	30.5	30.5×10^3
			5	1000	1729.5	1793.5×10^3
			6	1000	42.5	42.5×10^3
			7	2000	451.5	903.0×10^3
			8	2000	24.0	48.0×10^3
			9	1000	2756.5	2756.5×10^3
						5608×10^3

10,000 Transactions Breakdown by %

Update Time		Query Time			
Percent	Time in Accesses	Percent	Time in Accesses	Total Time	Performance
100	554×10^3	0	0	554×10^3	2.9
75	414×10^3	25	1402×10^3	1615×10^3	2.3
50	277×10^3	50	2803×10^3	3080×10^3	2.2
25	138×10^3	75	4205×10^3	4342×10^3	2.18
0	0	100	5608×10^3	5607×10^3	2.19

SECTION VI

PATTERN CLASSIFICATION

PROBLEM STATEMENT

A pattern is defined to be a row vector of dimension 1 x t (t unspecified) whose components are real numbers. One subclass is singled out for specia attention—those vectors whose components are either zeroes or ones.

This definition of a pattern has been adopted because it includes as a sp d case those patterns which are defined as matrix or tabular arrays. If a patt n is given as a matrix composed of, say, m rows and n columns, then such a matrix can be rewritten as a row vector of dimension 1 x (mn) by placing th rows (or the columns) consecutively from left to right. Thus only row vect need be considered.

The chief goals of pattern analysis are:

1. To cluster the patterns into similar groups.
2. To assign new patterns to the appropriate groups by measuring th similarity between the patterns and the groups.

In order to achieve these goals, it is evident that several basic proble must be solved.

- The concept of similarity of patterns must be defined as well as t concept of a pattern cluster or pattern group.
- Techniques must be developed for measuring the similarity betwe wo patterns and the similarity between a pattern and a cluster of oth patterns.
- Decision rules must be formulated for assigning new patterns to appropriate clusters.

CHARACTERISTICS OF PROBLEM

Having defined a pattern as a vector, it becomes clear that the significance of a pattern depends on either:

a. The magnitude of the entries in the pattern

b. The sequence in which a critical set of values occur

c. A combination of (a) and (b), i.e., when magnitude and sequence must be considered together.

It will simplify the presentation if (a), (b), and (c) are discussed separately. With respect to (a), a further simplification will be made. Initially only vectors of zeroes and ones will be examined. This may be denoted as case (a. 1).

For (a. 1) then, the problem may be expressed as follows: given a set of p patterns, each of which is a 1 x t vector of zeroes and ones, find meaningful relationships among the patterns. Usually, "meaningful relationships" will mean: "cluster the patterns into similar groups".

At this point, it will be useful to introduce some notation and definitions. Assume that the patterns have been aligned one under the other, thereby forming a p x t matrix, $R = (r_{ij})$, where $i = 1, \ldots p$ and $j = 1, \ldots, t$.

Let $r_{ij} = 1$ is pattern i has a one in position j.

Let $r_{ij} = 0$ otherwise.

Define $R_i = \sum_{j=1}^{t} r_{ij}$. R_i is the number of ones in pattern i. R_i will be called the sum function for pattern i.

Define $A_{ik} = \sum_{j=1}^{t} r_{ij} r_{kj}$. A_{ik} is the number of positions in patterns i and k which have matching ones. It is evident that the product $r_{ij} \cdot r_{kj}$ is one only when both r_{ij} and r_{kj} are one. Summing over j thus counts the number of places where patterns i and k have corresponding ones. A_{ik} is the intersection function for patterns i and k.

Define $V_{ik} = R_i + R_k - A_{ik}$. V_{ik} is the number of positions in which there is a one in either pattern i or pattern k or both. V_{ik} is the union or inclusive-or function for patterns i and k.

Define $H_{ik} = R_i + R_k - 2A_{ik}$. H_{ik} is known as the Hamming distance—the number of non-matching positions in patterns i and k, i.e., H_{ik} is the number of positions in which there is a one in either pattern i or pattern k but not in both. H_{ik} is the exclusive-or function for patterns i and k.

It should be noted that all the above functions are concerned with the number of ones and zeroes in a pattern but not with the arrangement or sequence of ones and zeroes.

Returning now to the problem of clustering patterns, it is clear that the definition of the term "similar patterns" is the key to the problem. Previous work in this field has shown that the definition of similarity depends heavily on the particular problem being investigated. For example, in reference (1) below, five definitions of similarity are described. Letting S_{ik} denote the measure of similarity between patterns i and k, then in terms of the notation just defined, the five measures of similarity described in (1) are:

1. King - Tanimoto:

   $S_{ik} = A_{ik}/V_{ik} = A_{ik}/(R_i + R_k - A_{ik})$.

2. Baxendale:

   $S_{ik} = A_{ik}/t$.

3. Kochen - Wong:

   $$S_{ik} = \frac{tA_{ik}}{R_i R_k}$$

---

1. Applied Research Program, AIDS, Quarterly Report No. 3, volume 1 - Progress Summary, Feb. 28, 1962, IBM Research Center, Yorktown Heights, New York.

4. Luhn - Savage:

$$S_{ik} = \frac{A_{ik}}{R_k}$$

5. Stiles:

$$S_{ik} = \frac{\log_{10} t\ (tA_{ik} - R_i R_k - \frac{t}{2})^2}{R_i R_k (t-R_i)(t-R_k)}\ .$$

Examination of these five measures of similarity reveals that:

1. They are independent—none is a function of any other
2. Each definition is some combination of the four numbers: t, R_i, R_k and A_{ik}.

Since V_{ik} and H_{ik} are also expressible in terms of t, R_i, R_k and A_{ik}, it is evident that the computation of these four numbers is basic to any pattern analysis program. Accordingly, the ability to calculate rapidly the [illegible]n and the intersection functions is put forth as a basic requirement.

On the other hand, precisely because these are so many definitions of similarity, it appears that the computation of the sum and intersection functions together with t represents the extent of basic techniques which will be practical for a general pattern analysis program. This concludes the discussion of case (a. 1).

Turning now to the examination of case (b), it will be useful once again to limit the discussion to 0 - 1 vectors. For case (b), the problem of pattern analysis becomes more complex than for case (a). Not only is the number of ones significant but in addition the location or arrangement of the ones (and consequently the arrangement of the zeroes) must be taken into account. Consider the simple situation depicted in Figure 20 for example. Let the 8 x 10 matrix be interpreted in the usual way as a 1 x 80 pattern. Then the character "7"

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	1	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Character "7" is defined as the set of ten ones located in positions 19, 20, 21, 22, 27, 30, 38, 46, 54, 62.

Figure 20. Pattern for the Character "7"

is defined not simply as a set of ten ones but as the set of ten ones located in the specific positions: 19, 20, 21, 22, 27, 30, 38, 46, 54 and 62. It is evident that the arrangement of ones defines a character. Identical characters are those whose defining sequences are the same. Further, if two defining arrangements are such that they have the same number of ones and if all corresponding sequence positions differ by the same constant, then one sequence represents a shift left or right, up or down of the character defined by the second sequence. For example, the sequence 1, 2, 3, 4, 9, 12, 20, 28, 36, 44 differs in each place by 18 from the sequence defining "7". Accordingly, the new sequence represents "7" shifted up two and two places to the left.

In many practical problems, sequences to be compared will not have the same number of ones. In such cases, a useful procedure may be to compare the longer sequences against the shortest sequence of interest, since often the basic problem is to determine those patterns which contain the shortest sequence as a subsequence. Consider, for example, pattern A, B and C which are defined as twenty position vectors:

A: 10011100101101001011

B: 00000011111001011001

C: 01001001100111010110

Suppose the problem is to find all patterns which contain pattern G as a subsequence where G is a ten position vector:

G: 1110010110.

Comparison of G with A, B and C shows that G matches positions 4 through 13 of A, positions 9 through 18 of B and does not match at all with C. (In a computer with an associative memory capability this matching will probably be most

easily accomplished by the "match and link right" techniques previously proposed for the text searching and term identification problem.)

In order to speed up the matching process, it is proposed that the data be compressed by use of the following information preserving data compression scheme. Let the 0 - 1 vector which defines pattern A be replaced by another vector, V(A), whose components are the distances between the consecutive ones in pattern A. This defines the components of V(A) from the second component on. To complete the definition, the first component in V(A) will be defined as the position of the first "1" in A, i.e., if the first "1" in A occurs at position 3 in A, the first component in V(A) is 3. For example, the compressed vectors associated with patterns A, B, C and G are:

V(A) = 13113212321

V(B) = 711113213

V(C) = 2331311221

V(G) = 111321

It is important to note that since the first component of a vector V(P) marks the position of the first "1" in P, it is therefore possible to reconstruct P given V(P). Also note that the sum of the components in V(P) which can be denoted as SV(P) is equal to the position number of the <u>last</u> "1" in P. This fact can be used to provide a rough error check when forming vector V(P) from pattern P.

If the problem of finding all patterns which contain G as a subsequence is reworked using the vector patterns, it is clear that the technique of "match and link right" can again be used. The comparison time will be shorter since the vectors are shorter. However, in using V(G) instead of G, a problem arises from the fact that the last component of V(G) refers only to the distance between

the last two ones in G. The fact that G might have ended in a string of zeroes is not indicated in V(G). The same is true of V(A), V(B) and V(C). Consequently, the vector definition for pattern P should be slightly modified so as to indicate how P ends. The modification is very simple. Let the new final component of V(P) denoted as LV(P) be the difference between the number of positions in P and the position of the last one in P. Recalling that the position of the last one in P is the sum of the components in V(P), excluding LV(P) of course, it is clear that if pattern P has t positions, then LV(P) = t-SV(P). LV(P) will be zero if P ends in a one and will be a positive integer otherwise. With this change, the vectors for A, B, C and G become:

V(A) = 131132123210

V(B) = 7111132130

V(C) = 23313112211

V(G) = 1113211

In matching V(G) with the other vectors, a match on first components will occur provided the first component of V(G) is less than or equal in value to the component of the other vector with which it is being matched. This follows because of the fact that the first component simply indicates where in G the first "1" is located. Note also that the last component, LV(P), matches whenever it is strictly less than the component with which it is being compared.

A comparison of V(G) with V(A) shows that V(A) contains the sequence 3113212 in positions 2 through 8. There is perfect agreement between V(G) and this sequence from the second through the penultimate positions. The first components match since 1 is less than 3. Finally the last component of the subsequence being greater than LV(G), there is agreement in the last place also. Accordingly, G is a subsequence of A.

Examination shows that V(B) contains the subsequence 1113213. Since the last component of this subsequence is larger than LV(G), there is agreement between the subsequence and V(G).

Finally examination of V(C) shows that the closest matching sequence to V(G) is 3112211.

In reviewing case (b. 1), it is evident that attempting to find subsequences which match exactly a given sequence or sequences is only one of several possible approaches to the problem of determining similarity based on arrangement of ones and zeroes. However this approach appears to be such a basic procedure that it should be incorporated as a basic tool into any general pattern analysis program. This concludes the discussion of case (b. 1).

It will be recalled that case (a. 1) considered vectors of zeroes and ones. A slightly more complex situation will be considered now in which the components are scaled values. In other words, a component value no longer is simply 0 or 1. Instead it is a number taken from some range of values. These scaling factors often are used to quantize non-numeric phenomena such as shadings (from light to dark) or distances (from near to far) or weights (light to heavy). Thus, the vectors are composed of integer components. By adding a constant, if necessary, to each component, these integer values can always be made strictly positive so that they all lie in some interval (a, b), where a and b are positive integers. It will be assumed that this has been done and further it will be assumed that the interval is from 1 to N, where N is known.

Case (a. 2) may now be expressed as: "Cluster patterns whose components are positive integer scaling factors taken from the range 1 to N." It is evident that in this case there is an obvious meaning to the notion of distance between

patterns or similarity between patterns. Patterns are similar if the differences between their respective components are small. For example the vector 71234 is closer to 61234 than it is to 71434 since it differs by 1 in one place from the first while it differs by 2 in one place from the second. But what about the distance between 71234 and 62234? Here the difference is one in each of two places. Clearly the problem is to develop some similarity measure based on the component differences. Since the components are scale factors, elaborate formulas should not be necessary. A minimum of computation is desired. Note that case (c) is essentially the same as case (a.2).

Consequently if the similarity measure is denoted as S_{ik} where i and k refer to pattern i and k, then S_{ik} will be some function of r_{ij}, and r_{kj}.
Examples of such a function are:

a. $S_1 = \sum_j \left| r_{ij} - r_{kj} \right|$ = the sum of the absolute values of the differences.

b. $S_2 = \sum_j \left(r_{ij} - r_{kj} \right)^2$ = the square of the Euclidean distance between vectors i and k.

c. $S_3 = \max_j \left\{ \left| r_{ij} - r_{kj} \right| \right\}$ = the largest single component distance between i and k.

d. The correlation coefficient denoted by r may also be considered a measure of estimate of the similarity between two patterns. Recall that the formula for r is:

$$r = \frac{t \sum_j r_{ij} r_{kj} - \sum_j r_{ij} \sum_j r_{kj}}{\left[\left(t \sum_j r_{ij}^2 - \left(\sum_j r_{kj} \right)^2 \right) \left(t \sum_j r_{kj}^2 - \left(\sum_j r_{kj} \right)^2 \right) \right]^{1/2}}$$

It is clear that r requires much more computation than the other measures. However, its calculation would be facilitated if the <u>inner product</u> function of two

vectors could be calculated rapidly, for it is evident that $\sum r_{ij}^2$ may be regarded as the inner product of a vector with itself, while $\sum r_{ij}$ may be considered as the inner product of a vector with a vector of ones and $\sum r_{ij} r_{kj}$ is the regular inner product of i and k. Consequently, the correlation coefficient may be interpreted as combination of various inner products.

e. A rather extreme example of a measure is the outer product of two vectors. In geometric terms, the outer product is a vector whose length is numerically equal to the area of the parallelogram formed by the two vectors and whose direction is perpendicular to the plane of the two vectors. If $Y_{ij} = (y_1, \ldots, y_t)$ denotes the outer product of patterns i and k, then an important relation among the components y_j, r_{ij} and r_{kj} is given by the Lagrange Identity:

$$\sum_j y_j^2 = \left(\sum_j r_{ij}^2\right)\left(\sum_j r_{kj}^2\right) - \left(\sum_j r_{ij} r_{kj}\right)^2$$

As in the case of the correlation coefficient, it is interesting to observe that the right hand side is a combination of various inner products. Again, the need for an inner product function is indicated.

A third reason for developing inner product function capability is that the Euclidean distance may be expressed in terms of inner products:

$$\sum_j \left(r_{ij} - r_{kj}\right)^2 = \sum_j r_{ij}^2 + \sum_j r_{kj}^2 - 2\sum_j r_{ij} r_{kj}$$
$$= \sum_j \left(r_{ij}^2 + r_{kj}^2 - 2r_{ij} r_{kj}\right).$$

In summary then, analysis of the characteristics of the pattern classification problem leads to recommending the following as design requirements:

1. The ability to calculate rapidly the sum and intersection functions of 0 - 1 type vectors;

2. The ability to find rapidly those subsequences of 0 - 1 type vectors which match exactly one or more given sequences;

3. The ability to compute rapidly the inner product function of two vectors having positive integers as components.

Finally it should be remarked that requirement (3) is most fundamental since (1) and (2) may be regarded as special cases of the inner product function.

PROCESSOR DESIGN

As the analysis of the pattern finding and classification problem indicates, the inner-product notion of vector algebra is the basic tool used in the great majority of the pattern manipulation schemes. This function is computed in "ordinary" computers by bringing out in sequence each component of the vector and performing the operations necessary. This is done for each pattern model against which the incoming pattern vector is to be matched. The most significant gain, then, can be made in this area and this design is directed to taking full advantage of any savings possible in parallel operations in this respect.

The computer which is depicted in Figure 21 is divided into three sections. These are:

1. Program memory and arithmetic-logic unit
2. Main parallel store
3. Small parallel store

The program memory is a conventionally organized memory with random access and sequential instruction read out. Program control branching is available for program modifications, etc. The arithmetic-logic unit is also conventional in nature and capable of data transfers, arithmetic operations, and other similar control functions. The memory is logically divided into an instruction section and a data section. The instruction section has registers which are one half or one quarter as long as those in the data section. The data section registers are the same length as the main and small memories, that is, long enough to contain an eight or ten position vector with integral values. This arrangement may be obtained via the multiple read out technique of System/360 or by using the memory cells directly.

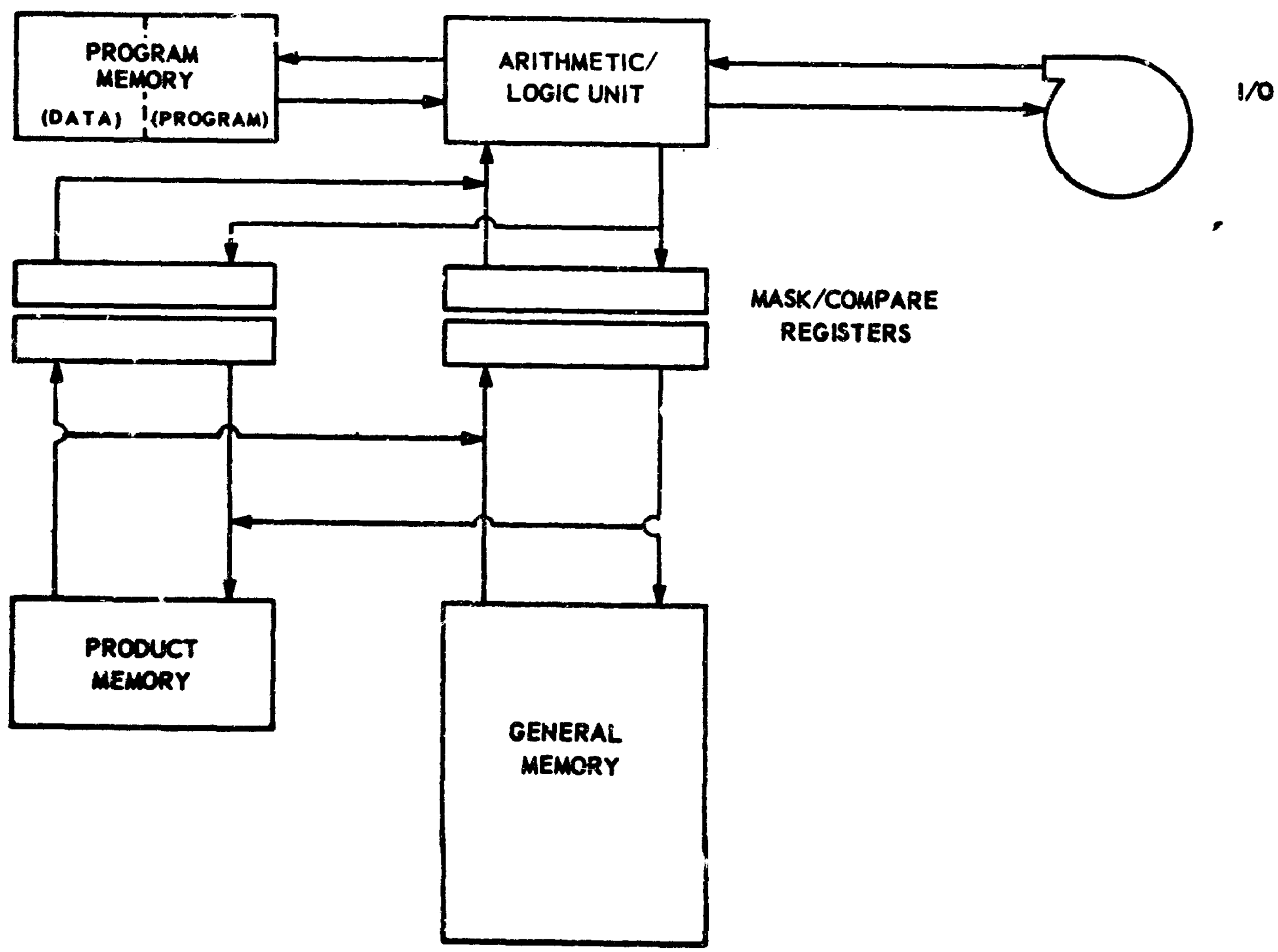


Figure 21. Processor for Pattern Classification

The main parallel store is fully associative memory. The usual instructions for an associative processor apply. A list of such usual instructions may be found in reference (2), below. There are some additional instructions described in Appendix I which apply to this memory. Each register of this memory will contain, as in the program memory, eight or ten positions. In addition, the registers will carry an eight bit counter which will also be associatively addressable. The arithmetic-logic unit will be able to add to the counter directly, modified by the presence or absence of a match bit. Link bits and match bits will be available to allow for the interconnection of several of these counters from consecutive registers. By suitable combination of link bits one may make the counter (counters connected) appear as an adder with either end-around carries or in which carries out the high order position are lost. Accumulation may be carried several registers ahead so that sums from various parts of a vector may be accumulated into one register.

The small store is also fully associative but has only a few registers. There is an accumulator register associated with each storage position of this memory. When the inner product function is performed against the memory, the resultant sum is accumulated in it. The ordinary associative processes apply to both the storage register and to the associated accumulator register. All other of the special instructions apply with the exception of the Intersect operator.

The following brief, simple example demonstrates some of the considerations that will influence the computation of the various distance measures.

---

2. Estrin, G. and Fuller, R., "Algorithms for Content Addressable Memories," Proceedings of the Pacific Computer Conference, Pasadena, California, March 15, 16, 1963.

In the case of 0–1 vectors, the inner product function yields the clue to group membership. For example, in the Tanimoto scheme, the measure is $d(v, v_j) = \frac{A_j}{R + R_j - A_j}$; for any threshold value S, if the requirement is that

$\frac{A_j}{R + R_j - A_j}$ must exceed S, then by successively setting A_j equal to R_j and then to R and solving the inequality $A_j/(R + R_j - A_j) > S$ for R_j, it follows that the admissible range of values of R_j is given by:

$$\left\{R_j\right\} = \left\{r \,\middle|\, sR \leq r \leq R/x\right\}.$$

For this set of values of R_j only those ones are to be taken whose values A_j are maximal and the comparison will be among just those. The total number of comparisons in this case will be greatly reduced being just that number equal to the number of integers contained in the range $sR \leq I \leq R/s$. The search may then be further limited by calculation as to the possible values of A_j that could possibly cause other values of R_j to exceed the preciously obtained maximum. Similar strategies may be used in the case of other measures of distance.

Finally, formulas can be derived for estimate of the performance of the processor. To be definite, the computation of the inner product function is considered, first by using the small store and then by using the main store. The integer case and the 0-1 case are considered separately.

a. For the integer case, let:

N = number of registers in the small store

T = number of integers per small store register

X = number of patterns

K = number of integers per pattern

a = number of processor machine cycles required for the integer inner product function using the small store

b = number of conventional machine cycles required for an accumulative multiply operation.

Furthermore, let the notation $\lceil X \rceil$ represent an integer $K = \lceil X \rceil$ such that $K - 1 < X \leq K$. Then the number of cycles I_p required by the processor for the integer inner product computation is given by:

$$I_p = a \left\lceil \frac{X \left\lceil \frac{K}{T} \right\rceil}{N} \right\rceil + X \left\lceil \frac{K}{T} \right\rceil$$

The number of cycles I_c required by a conventional machine for the integer inner product computation is given by

$$I_c = bKX.$$

If the cycle time of both machines is assumed the same, and if:

$N = 32$
$T = 10$
$X = 40$
$K = 15$
$a = 5$
$b = 5$

then the performance ratio $R = I_c/I_p$ is

$$R = \frac{3000}{95} = 32.$$

b. For the 0 - 1 case, let:

T = number of bits per conventional register

K = number of bits per pattern

X = number of patterns

a = number of processor machine cycles required for the 0 - 1 inner product function using the main store

b = number of conventional machine cycles required to count the number of one bits in a register

Then the number of cycles L_p required by the processor for the 0 - 1 inner product computation is given by:

$$L_p = a$$

The number of cycles L_c required by a conventional machine for the 0 - 1 inner product computation is:

$$L_c = (b + 1) X \left\lceil \frac{K}{T} \right\rceil$$

If the cycle time of both machines is assumed the same, and if:

$$\begin{aligned} T &= 36 \\ K &= 120 \\ X &= 40 \\ a &= 50 \\ b &= 5 \end{aligned}$$

then the performance ratio $R = L_c/L_p$ is:

$$R = \frac{960}{50} \approx 19.$$

EXAMPLE PROBLEM

In order to demonstrate the formation of pattern clusters and the assignment of patterns to clusters, an experiment was performed using simulated data. The procedures that would be used in carrying this experiment on a conventional computer and on an associative processor were both manually simulated, thereby yielding a comparison with the number of operations that would be required in the conventional computer with the number that would be required in the

computer having associative features. Since the experiment was manually performed, the patterns were restricted to integers. This restriction minimized computational difficulties yet still permitted a valid simulation of the general case since an integer may be treated as a row vector of dimension one by one.

The following sections a through f describe the experiment fully:

a - Definitions and notation

b - Decision rules for assigning patterns to clusters

c - Determination of nodal values

d - Experimental procedure

e - Description of pattern data and resulting clusters

f - Conclusions

a. Definitions & Notation

A pattern element (P) is any integer in the range 0 to 99 inclusive, i.e. the pattern is a row vector of dimension one by one.

A cluster (C) is any non-null set of pattern elements which meets the rules of membership.

A node (N) of a cluster is that integer which is closest to the arithmetic mean of the pattern elements of the cluster. (Round down at the fractional part equal 0.5).

The distance d (C, P) between a cluster C and a pattern element P is the absolute value of the difference N-P, where N is the node of C. Thus, $d(C, P) = \left| N_C - P \right|$.

b. Decision Rules for Assigning Patterns to Clusters

Rule 1. Assign the first P as the first C.

Rule 2. Assign subsequent P's to C_i whenever d (C_i, P) $\leq$ K and d (C_j, P) $>$ K for all j $\neq$ i. K is some cutoff value defined by the user.

Rule 3. Assign P to C_i when d (C_i, P) $<$ d (C_j, P) and d (C_i, P) $\leq$ K.

Rule 4. Assign P to C_i when d (C_i, P) = d (C_j, P) $\leq$ K and when the number of elements of C_i is less than the number of elements of C_j.

Rule 5. Assign P to C_i and to C_j whenever d (C_i, P) = d (C_j, P) $\leq$ K and the number of elements of C_i equals that of C_j.

Rule 6. Assign P as a new C whenever d (C_i, P) $>$ K for all i.

c. Determination of Nodal Values

The node of a cluster will initially equal its first member. The node will thereafter be the average of the cluster elements whenever the number of elements in the cluster is congruent to 0 modulo 10. This yields results which do not significantly differ from those obtained by re-evaluating after the addition of each member. A final determination will be made after the last elements of the data sample has been assigned.

d. Experimental Procedure

The procedure begins with N_1 = P_1 and $P_1 \in C_1$. N_2 will be the first P_i such that d (C_1, P_i) $>$ K. When all P_i have been assigned to C_j, the N_j are re-calculated. As a result of this computation, the final N_j may be such that for some $P_i \in C_j$, d (C_j, P_i) $>$ K. Therefore, the pattern elements P_i need to be re-evaluated. Further, since the C_k that are of interest are only those with a number of elements deviating from and significantly higher than the average

number, only those C_k where the number of elements equals or exceeds the average by at least 20% will be considered. A new determination is made using just those values as nodal values by the same assignment algorithm. A change is made at this time in the assignment rules:

Rule 1. * $P_i \in C_j$ when $d(C_j, P_i) < K$. When $d(C_{j1}, P_i) < K$ and $d(C_{j2}, P_i) < K$, assign P_i to C_{j1} and C_{j2}.

Rule 2. * Ignore all P_i such that $d(C_j, P_i) > K$ for all j.

This pass is repeated until no further changes occur in the clusters or their elements. By this technique it is seen that when the true nodal value falls between two clusters (by chance of ordering the original pattern), the two clusters will coalesce into one cluster.

e. Description of Pattern Data and Resulting Clusters

For the experiment, 100 simulated pattern values were obtained from a random normal number table. The 100 numbers are listed in Table XXVII and are divided into two groups. The first group was centered about 30 with a standard deviation of 10; the second group was centered about 60 with a standard deviation of 10. Then the numbers were scrambled as shown in Table XXVIII by the rule: start with column 1 of this first group and transcribe the number. If the number is even, continue with the first group. If the number is odd, take the next number from column 6-- the first column of the second group. If the number from the second group is even, continue with the second group. If the number is odd, take the next number from the first group. Repeat until all the numbers have been transcribed.

The experiment was performed three times with K equal to 8, 10 and 12. The results of the first pass are shown in Tables XXIX, XXX and XXXI while the results for the second pass are shown in Tables XXXII, XXXIII and XXXIV. Seven clusters were formed for pass one at width K=8 and K=10 while six were formed at width K=12. Table XXXV summarizes the results of pass one, displaying the nodal values and the number of elements in each cluster for widths, 8, 10 and 12 respectively. Table XXXVI displays similar information for the results of pass two.

Examination of Table XXIX indicates that clusters 3 and 6 for width K=8 exceed by more than 20% the value of 14 to 15 members per cluster that would be expected if the 100 patterns were uniformly distributed over the 7 clusters. Accordingly on pass 2 for K=8, the 100 patterns are matched against the nodal values of 32 and 58 and are assigned according to Rules 1* and 2*. Similar procedures are followed for re-evaluating the clusters for widths K=10 and K=12.

On pass two then, the number of clusters was reduced to two, three and two for K equal to 8, 10 and 12, respectively. It will be noted that the clusters formed have nodes quite close to the original values with the exception of the second cluster formed for width 10. However this latter cluster has significantly fewer elements than either of the other two clusters.

Column A of Table XXXVII shows the number of operations that would be performed in a computer with a conventional memory in carrying out this experiment and Column B shows the number that would be performed using a computer with an associative memory.

Before comparing this data, it will be useful to explain how the comparison figures relevant to the computer with the conventional memory were calculated.

The figures given in Column A for the total number of comparisons required for each pass and each width using a computer with a conventional memory were all arrived at by comparing each new pattern against the total number of clusters that has been formed up to that time and assigning the pattern according to the decision rules. In order to demonstrate this procedure, the figures of 645 comparisons for pass 1 with K=8 and of 562 comparisons for pass 1 with K=10 will be explained in some detail.

For pass 1 with K=8, the patterns in Table XXVIII are examined row by row, i.e. from left to right, top to bottom. Accordingly, 19 becomes the first member of and the nodal value of cluster 1. 6 is matched against 19, is found to be too far away and so becomes cluster 2. 30 is matched against clusters 1 and 2 and becomes cluster 3. 42 is matched against clusters 1, 2 and 3 and becomes cluster 4. Continuing with row 1, 18 is matched against the four clusters and is assigned to cluster 1, 35 is assigned to cluster 3, 71

becomes cluster 5 after being matched against the four clusters, 56 becomes cluster 6, 56 is assigned to cluster 6 and the last member of the first row, 69, is assigned to cluster 5. Thus the number of comparisons required to assign all the members of the first row may be evaluated as follows:

row element	no. of comparisons required
19	0 (begins cluster 1)
6	1 (begins cluster 2)
30	2 (begins cluster 3)
42	3 (begins cluster 4)
18	4
35	4
71	4 (begins cluster 5)
56	5 (begins cluster 6)
56	6
69	6
	35 = total for row 1

Continuing with row 2, it is found that all members of the second row are assigned to the six existing clusters. Thus row 2 requires 10 x 6 or 60 comparisons. In the same fashion, it is found that all members of row 3 are assigned after six comparisons each. However the last member of row 3, 81, becomes the first member of cluster 7. This turns out to be the last cluster formed. The

members of the remaining seven rows are all assigned to these seven clusters after seven comparisons each. Thus the total number of comparisons required for pass 1 with K=8 is:

row	no. of comparisons required
1	35
2	60
3	60
4	70
5	70
6	70
7	70
8	70
9	70
10	70
	645 = total number of

comparisons for pass 1 with K=8.

For pass 1 with K=10, the patterns in Table XXVIII are examined column by column, i.e., top to bottom, left to right. Thus, 19 becomes cluster 1, 57 becomes cluster 2, 18 is assigned to cluster 1, 30 becomes cluster 3, 34 and 31 are assigned to cluster 3, 62 is assigned to cluster 2, 58 is assigned to cluster 2, 75 becomes cluster 4 and the last member of column 1, 28, is

assigned to cluster 3. Thus for column 1, the number of comparisons required is:

column element	no. of comparisons required
19	0 (begins cluster 1)
57	1 (begins cluster 2)
18	2
30	2 (begins cluster 3)
34	3
31	3
62	3
58	3
75	3 (begins cluster 4)
28	4
	24 = total for column 1

Continuing with column 2, it is found that the first element of column 2, 6, becomes cluster 5 and the fourth element 44, becomes cluster 6. The remaining elements of column 2 and the members of of columns 3 through 10 are assigned to the six clusters after six comparisons each until the seventh element of column 10, 87, is reached, which becomes cluster 7. The last three elements of column 10 thus require seven comparisons each. The total

number of comparisons required for pass 1 with K=10 is therefore calculated to be:

column	no. on comparisons required
1	24
2	55
3	60
4	60
5	60
6	60
7	60
8	60
9	60
10	63
	562 = total number of

comparisons for pass 1 with K=10.

For pass 1 with K=12, the data in Table XXVIII is again examined column by column yielding a total of 475 comparisons.

CONCLUSIONS

For the data in column B of Table XXXVII it is evident that the number of comparisons is equal to the number of items examined. Examination of and comparison of Columns A and B of the table leads to the following conclusions.

(1) In a computer with an associative memory capability, the number of comparisons that must be performed for any pass

depends strictly on the number of items that are examined. The number of comparisons is independent of the number of clusters formed during that pass.

(2) In a computer with a conventional memory the number of comparisons that must be performed depends on both the number of items to be examined and the number of clusters formed. Roughly speaking, the number of comparisons for pass 1 is very nearly equal to the product of the number of items examined times the number of clusters formed. For pass 2 and succeeding passes, the number of comparisons needed is exactly equal to the number of items times the number of clusters.

(3) From (1) and (2), it is evident that the associative memory capability always effects a reduction in the number of comparisons that mus be performed. It is further evident that the ratio of improvement will increase as the number of clusters formed increases.

TABLE XXVII. TWO SETS OF PATTERN VALUES

First Group					Second Group				
19	40	34	35	31	57	76	69	51	51
18	17	42	49	26	62	68	57	56	47
30	30	24	39	31	50	31	64	30	60
34	32	25	10	33	75	59	55	53	69
31	31	26	45	33	60	67	60	69	81
28	38	18	43	19	44	71	71	60	59
6	30	34	23	25	69	78	48	63	81
33	42	21	24	9	63	64	42	56	53
42	32	42	21	22	60	58	53	47	63
24	33	35	44	11	60	57	80	69	49

$\bar{x} = 29.3$ $\bar{y} = 59.9$

TABLE XXVIII. THE FIRST AND SECOND GROUPS SCRAMBLED

[illegible]	6	30	42	18	35	71	56	56	69
57	33	32	32	34	69	43	30	47	9
18	60	31	33	21	49	48	53	33	81
30	44	60	59	71	57	42	44	69	22
34	69	60	34	42	39	53	31	33	11
31	42	76	42	35	64	23	69	61	59
62	24	68	24	78	55	80	60	19	87
58	40	31	25	84	10	51	63	47	53
75	17	36	67	58	45	24	26	25	63
28	63	30	26	57	60	21	31	60	49

TABLE XXIX. FORMATION OF CLUSTERS WITH K=8

PASS 1

Cluster:

1		2	3			4		5		6			7
19	17	6	30	30	31	42	42	71	67	56	53	63	81
18	24	9	35	34		43	40	69		56	64	53	78
18	26	11	33	34		47	47	69		57	61	58	80
21	25	10	32	31		48	38	71		60	59	63	87
22	26		32	33		44	45	69		49	62	63	
23	21		34	31		42	49	69		53	55	57	
24			30	35		44		76		60	60	60	
24			31	31		42		69		59	58	60	
19			33	28		39		68		57	64		
25			33	20		42		75		60	51		

TABLE XXX. FORMATION OF CLUSTERS WITH K=10

PASS 1

Cluster:

1		2			3			4		5	6		7
19	21	57	57	63	30	32	31	75	69	6	44	43	87
18	19	62	57	56	34	33	33	69	69	10	42	48	
24	25	58	64	61	31	34	33	76	69	9	40	42	
17	22	60	55	60	28	26		68	81	11	38	44	
24		63	60	59	33	34		67			42	47	
25		60	53	53	30	35		71			42	47	
18		60	51	63	32	35		78			42	49	
21		59	56		31	30		69			49		
23		64	53		31	31		71			39		
24		58	60		30	26		80			45		

TABLE XXXI. FORMATION OF CLUSTERS WITH K= 12

PASS 1

Cluster:

1		2			3				4		5	6
19	23	57	57	53	34	31	35	47	75	69	6	87
18	24	62	49	60	31	38	35	33	69	69	10	
30	21	58	57	63	28	30	39	33	76	69	9	
24	26	60	64	56	33	42	45	47	68	81	11	
17	19	63	55	61	44	32	43		67			
24	25	60	60	60	42	33	42		71			
25	22	60	48	59	40	34	30		78			
26		59	53	53	30	42	44		69			
18		64	51	63	32	34	31		71			
21		58	56	49	31	42	31		80			

TABLE XXXII. FORMATION OF CLUSTERS WITH K=8

PASS 2

Cluster:

1				2		
30	31	35	25	57	57	63
34	38	35		62	57	56
31	30	39		58	64	61
28	32	24		60	55	60
33	33	30		63	60	59
24	34	31		60	53	63
40	24	26		60	51	53
30	25	31		59	56	
32	26	33		64	53	
31	34	33		58	60	

TABLE XXXIII. FORMATION OF CLUSTERS WITH K= 10

PASS 2

Cluster:

1				2		3			
30	31	34	31	34	35	57	67	53	60
34	38	21	33	44	35	62	64	51	69
31	30	35	25	42	49	58	58	56	59
28	32	35	22	40	39	60	57	53	53
33	33	39		38	45	69	69	69	63
24	34	23		42	43	63	49	60	49
40	24	24		34	48	60	57	63	
30	25	21		42	42	60	64	56	
32	26	31		34	53	68	55	69	
31	30	26		42	51	59	60	61	

TABLE XXXIV. FORMATION OF CLUSTERS WITH K=

PASS 2

Cluster:

		1				2		
30	32	24	48	33	57	67		69
34	31	25	42	47	62	64		61
31	31	26	24	25	58	58		47
28	38	34	30		60	57		6[illegible]
33	30	42	44		69	69		69
44	42	35	31		63	57		59
42	32	35	26		60	64		5[illegible]
24	33	39	31		60	64		6[illegible]
40	34	45	47		68	55		4[illegible]
30	42	43	33		59	60		

TABLE XXXV. SUMMARY OF RESULTS OF PASS 1

Width K=8			Width K=10			Width K=12		
Cluster No.	No. of Elements	Nodal Value	Cluster No.	No. of Elements	Nodal Value	Cluster No.	No. of Elements	Nodal Value
1	16	20	1	14	21	1	17	22
2	4	9	2	27	59	2	30	58
3	21	32	3	23	31	3	34	36
4	16	43	4	14	72	4	14	72
5	11	70	5	4	9	5	4	9
6	28	58	6	17	44	6	1	87
7	4	81	7	1	87	-	-	-

TABLE XXXVI. SUMMARY OF RESULTS OF PASS 2

Width K = 8			Width K = 10			Width K = 12		
Cluster No.	No. of Elements	Nodal Value	Cluster No.	No. of Elements	Nodal Value	Cluster No.	No. of Elements	Nodal Value
1	31	31	1	34	30	1	43	35
2	27	59	2	20	42	2	39	59
-	-	-	3	36	60	-	-	-

TABLE XXXVII. SUMMARY OF COMPARISONS

	A	B
Pass 1, K=8		
Number of Patterns Examined	100	100
Number of Clusters Found	7	7
Number of Comparisons Required	645	100
Patterns/Cluster (Average)	14.3	14.3
Comparisons/Pattern (Average)	6.45	1.0
Pass 2, K=8		
Number of Patterns Examined	100	100
Number of Clusters Found	2	2
Number of Comparisons Required	200	100
Patterns/Cluster (Average)	-	-
Comparisons/Pattern (Average)	2.0	1.00
Pass 1 + Pass 2, K=8		
Number of Patterns Examined	100	100
Total Number of Comparisons Required	845	200
Comparisons/Pattern (Average) (C(A) and C(B))	8.45 C(A)	2.00 C(B)
Ratio of C(A) to C(B)	4.225 to 1.0	

TABLE XXXVII. SUMMARY OF COMPARISONS (Cont'd)

	A	B
Pass 1, K=10		
Number of Patterns Examined	100	100
Number of Clusters Found	7	7
Number of Comparisons Required	562	100
Patterns/Cluster (Average)	14.3	14.3
Comparisons/Pattern (Average)	5.62	1.0
Pass 2, K=10		
Number of Patterns Examined	100	100
Number of Clusters Found	3	3
Number of Comparisons Required	300	100
Patterns/Cluster (Average)	-	-
Comparisons/Pattern (Average)	3.0	1.0
Pass 1 + Pass 2, K=10		
Number of Patterns Examined	100	100
Total Number of Comparisons Required	862	200
Comparisons/Pattern (Average) C(A) and C(B)	8.62 C(A)	2.00 C(B)
Ratio of C(A) to C(B)	4.31 to 1.0	

TABLE XXXVII. SUMMARY OF COMPARISONS (Cont'd)

	A	B
Pass 1, K=12		
Number of Patterns Examined	100	100
Number of Clusters Found	6	6
Number of Comparisons Required	475	100
Patterns/Cluster (Average)	16.67	16.67
Comparisons/Pattern (Average)	4.75	1.0
Pass 2, K=12		
Number of Patterns Examined	100	100
Number of Clusters Found	2	2
Number of Comparisons Required	200	100
Patterns/Cluster (Average)	-	-
Comparisons/Pattern (Average)	2.0	1.0
Pass 1 + Pass 2, K=12		
Number of Patterns Examined	100	100
Total Number of Comparisons Required	675	200
Comparisons/Pattern (Average) C(A) and C(B)	6.75	2.0 C(B)
Ratio of C(A) to C(B)	3.375 to 1.0	

SECTION VII

MATHEMATICAL STUDIES

This section of the report discusses three studies that were conducted: a study of the possibility of developing a data distance measure; investigation of structured operation sets to be used in an associative processor; and a study of associative techniques for numeric processing. As might be inferred from their titles, these topics are somewhat peripheral with respect to the main design areas. However, they were selected because they offered promise of extending and improving associative processor capability.

Each of these three studies is self-contained and was conducted independently of the other two. None of the studies led to special processor designs (nor were any such designs intended). However, the results of the studies should provide supplementary guidelines to processor designers by supplying them with some simple, useful sets of operations for the processor.

DISTANCE MEASURE

In many military and commercial applications involving text processing, the probability of a significant error rate in the input is quite high. Error correction on present applications is performed manually or by complicated software at considerable expense. This study was undertaken to investigate lexical distance measures with potential for being implemented readily on associative processors.

In order to perform spelling correction, it will be necessary to compare words and to define in some meaningful way a distance between words. The definition of "distance between words" is non-trivial because words can differ by only one letter, yet their meanings may be completely different. For example, in the set (phone, shone, shore, short, shirt), each word differs by only one letter from its successor and predecessor, so that in one easily measured sense — number of common letters — these words are near each other. In a semantic sense, their distance is arbitrary since the meaning changes arbitrarily by substituting one letter for another. Consequently, word comparison techniques will be based either on semantic considerations or on mechanical features of a word but not on both. This study is concerned with developing distance measures based solely on such mechanical features as word length and spelling and is aimed at transcription and transmission errors, not content errors.

To begin with, attention is restricted to the following four causes of word errors:

1. Letters in the word are permuted or scrambled, e.g., COULDS, instead of CLOUDS
2. Correct order, but use of one or more incorrect letters, e.g., SEVIRIL instead of SEVERAL.
3. Too many letters, e.g., TELLEPHONE instead of TELEPHONE.
4. Missing letters, e.g., TRIFIC instead of TERRIFIC.

The selection of word errors as the ones to be analyzed was based on the results of an error study conducted by IBM under the 438L contract and reported in the working paper AIDS System Communication Error Study, 29 November 1961.

Each error cause is examined separately and some distance measurements based on these errors are developed. To aid in the analysis the following notation is used: n denotes the number of letters in the longer of the two words being compared. For errors 1 and 2, n is evidently the length of both words. The interchange of any two adjacent letters in a word is a transposition. Other notation is defined as appropriate.

Turning now to error 1, it is evident that a scrambled word may be unscrambled by a sequence of transpositions. It is known from group theory that this process is well defined and the number of transpositions required may be obtained by inspection of the permuted word. For a word n letters long, the maximum number of transpositions is required if the word is spelled backwards. This is obviously the worst case. Further, it is known that this maximum number is 1/2 (n) (n-1). If T is used to denote the number of transpositions actually required to unscramble a word, then one might consider as a distance measure the ratio

$$D(1) = T/\frac{1}{2}(n)(n-1).$$

D(1) has the advantage that its value lies between zero and one. However, it turns out that D(1) is not a very good measure, for in most cases, T will be either one or two thereby yielding an unreasonably small value for D(1). In other words since the number of transpositions, T, will never even approach

the value $\frac{1}{2}$ (n) (n-1) in actual cases, it is unreasonable to normalize T with respect to $\frac{1}{2}$ (n) (n-1). A more reasonable normalizing factor is n. Consequently, for error 1, let the distance between two words be given by Rule 1: $D(1a) = T/n$. Of course with this new definition, there now arises the possibility that T will be greater than n, yielding a distance greater than one. However this possibility is more theoretical than real, since as has already been stated, the distance will be of interest only when T is a fairly small number.

For example to demonstrate Rule 1, suppose the words are TRAVEL and TARVLE, and suppose that TARVLE is compared against TRAVEL. Each letter of TARVLE is examined to determine if it is in proper order with respect to the letters following it, i.e., the letters to the right of it. The examination shows that A is out of order with respect to R and L is out of order with respect to E. Hence A and L will each require one transposition so that a total of T=2 transpositions will permute TARVLE into TRAVEL. Since TRAVEL has six letters, it then follows that

$$D(1a) = 2/6 = 1/3.$$

For error 2 the occurrence of one or more incorrect letters, the obvious measure to use is to count the number of positions in which the characters disagree. In order to normalize this measure, the number of non-matching positions should then be divided by n, the length of the word. If K denotes the number of positions in which the two words disagree, then for error 2 the distance is given by Rule 2: $D(2) = K/n$. Evidently D(2) lies between zero and one. An example for rule 2 may be constructed by comparing SEPLRADE against SEPARATE. The two words disagree in the fourth and seventh positions so that K = 2. Since n = 8, D(2), = 2/8 = 1/4.

Errors 3 and 4, too many letters and missing letters, may be treated as one case by observing that both errors imply that words of unequal length are being compared. For error 3, the problem is the incorrect word has too many letters while for error 4, the situation is reversed — the incorrect word has too few letters. In either case, the following procedure for comparing words is applicable:

1. The words first are left justified. Regardless of which word is the correct one, the shorter word is completed to the length of the longer by adding null characters, labeled as *, to the end of the shorter word. Now both words are n characters long, where n is obviously the length of the longer word. Next, corresponding characters in the two are examined and a trial word $L = l_1 \, l_2 \, . \, . \, . \, . \, l_n$ is formed, where l_i is * if the two words disagree in position i and l_i is the common character, denoted as C_i, if the two words agree in the i-th position. Evidently the trailing characters of L must all be * since *'s were added to the end of the shorter word.

2. Next, right justify the two words and complete the shorter word to the length of the longer by adding *'s at the beginning of the shorter word. Again, examine the corresponding characters in the two words and form another trial word $R = r_1 \, r_2 \, . \, . \, . \, . \, r_n$ where r_i is * if the two words disagree in position i and r_i is the common character C_i if there is agreement in the i-th position.

3. Finally the trial words are compared with each other to produce a most probable word $P = P_i . \, . \, . \, P_n$. The comparison of L and R is performed

by "OR-ing" corresponding characters. Consequently it follows that

$P_i = C_i$ if either l_i is C_i or r_i is C_i or both are C_i;

$P_i = *$ if and only if both l_i and r_i are $*$.

It should be pointed out that there is no possibility that l_i and r_i can equal different characters. This may be shown by the following considerations. If l_i is C_i, this indicates that position i of the two words, left-justified, contains C_i. If r_i is C_i', then C_i' is in position i of the two words, right-justified. But the i-th position of the longer word left justified is the same as the i-th position of the longer word, right justified so that $C_i = C_i'$.

The distance between the two words is given by Rule 3: $D(3) = u/n$, where u is the number of *'s in the probable word P and n is the number of characters in P.

As one example of this technique, consider the comparison of COULDS and COLDS. First, left justify the two words and complete COLDS to a six character word by adding one * after S. Then COULDS and COLDS* yield L = CO****. Right justifying and adding one * before C leads to the matching of COULDS and *COLDS which produces R = ***LDS. Finally the "OR-ing" of L and R shows that P = CO*LDS. Hence by Rule 3, D(3) = 1/6.

A second example is given by comparing COAGE with COURAGE. Left justify and complete COAGE to the length of COURAGE by adding two *'s after E. Matching of COURAGE and COAGE** produces L = CO*****. Next, right justify and complete COAGE by adding two*'s ahead of the C. Matching of COURAGE and **COAGE produces R = ****AGE. Then the "OR-ing" of L and R yields P = CO**AGE, so that by Rule 3, D(3) = 2/7.

A few comments are in order regarding the formulation of Rule 3. At first it was planned to treat separately errors 3 and 4. However, a few test cases indicated that trying to distinguish between the problems of too many letters and too few letters was needlessly confusing. Development of a standard rule for comparing words of unequal length, without regard to which is the correct one, is the preferred approach.

To demonstrate these rules, consider the comparison of the test word COULDS against a list of correct words. COULDS will be said to match that word in the list to which it is closest according to any of the three distance measuring rules. The list of correct words in alphabetical order is given in the first column of Table XXXVIII. The second column states the rule which yielded the minimum distance. The third column lists the distances between COULDS and each of the correct words. From this column it is seen that COULDS is closest to the word COULD by Rule 3 and so will be said to match that word.

To accomplish these measurements in a conventional computer would require, generally speaking, that COULDS be compared against the correct words one at a time. On the other hand, an associative memory capability will permit the simultaneous comparison of the test word against the full list of correct words. Hence one may reasonably conclude that an associative memory capability reduces the number of comparisons required by a factor that is approximately equal to the number of correct words that are being matched against the test word.

TABLE XXXVIII. DISTANCE MEASURE EXAMPLE

Test Word: COULDS

Correct Word List (alphabetical order)	Rule use to yield minimum distance	Distance
BOARDS	2	1/2
CLOUDS	1a	1/3
COULD	3	1/6
COULEES	3	2/7
COULOMBS	3	3/8
COUNTESS	3	1/2
COUNTS	2	1/3
COWARDS	3	3/7
MOUNDS	2	1/3

COULDS is closest to COULD.

STRUCTURED OPERATION SET

This section of the report will discuss studies that were made during the investigation of the proposed development of a structured set of operations for the associative processor.

At the outset, a distinction will be drawn between the concepts of a structured set of operations and a structured operation code. The two ideas may contain overlapping features but neither necessarily includes the other. This investigation was concerned with the subject of a structured set of operations. However it will be useful to start by discussing briefly the concept of a structured operation code since this is perhaps the more familiar idea.

When one examines the operation instruction codes of the IBM 7094, one notices that the codes resemble those that might be used to structure an organization such as a manufacturing concern. That is to say, the codes represent a partition of the functions performed by the computer and the business respectively, as shown in Table XXXIX, but they are not organically or hierarchically related. Codes 03xx are neither inferior to nor subsets of codes 04xx. Since the codes are independent, it is not feasible to combine instructions simply by combining their respective codes in any meaningful way, i.e. one instruction followed by another usually will not produce the same result as an instruction whose numeric code is some logical or numeric combination of their respective codes.

However it should be noted that within one of the subsets of the partition, it may be feasible to consider a structured set of operations which would be applicable only within the subset. Referring once again to the IBM 7094,

examination of the 03xx codes indicates that most of them deal with the operations of addition and subtraction in the floating point mode. In addition, consideration is also given to the features of precision, normalization and signature. Thus altogether there are in the floating point mode, sixteen possible add-type instructions which are formed by taking all combinations of add or subtract, single or double precision, normalized or unnormalized, signed or unsigned, i.e. magnitude. A typical instruction is DUSM, octal code -0307; double precision, unnormalized, floating point subtract magnitude. By inspection of the codes 0300 through -0307, it is seen that all sixteen possible instructions are actually available and furthermore the unnormalized operations are characterized by having a minus sign in their octal codes. Thus one may conclude that there is a simple structure in this subset of operations, exhibiting the features of completeness and attribute characterization (at least for a single attribute).

Turning now to the problem of developing a structured set of operations, it would appear at first glance that some sort of hierarchic arrangement would be the most desirable and useful form for the structure to take. The reason for suggesting a hierarchic arrangement is that there are at least three levels of computer instructions which are so related: commands to open various gates, test triggers, etc. which are combined to form micro instructions; micro instructions which are put together to produce the so called machine instructions, and the machine instructions, which usually stand alone.

One example of a hierarchy is furnished by chemistry in which elements combine to form compounds. Further as shown in Table XXXX, an analogy can be drawn between the building blocks of chemistry and those of computers. As

is well known, the isotopes and the chemical compounds indicate by their formulas not only the elements from which they are formed but the proportion or amount of each element in the compound. Thus, $H_2 SO_4$ shows that one molecule of sulfuric acid is composed of two atoms of hydrogen, one of sulfur and four of oxygen. It is clear that one function of a structured operation set would be to obtain an analogous representation so that for example the code (or formula) for a machine instruction would make it possible to identify the micro-instructions which combine organically, so to speak, to form the machine instruction.

However it soon becomes clear that computer instructions are sufficiently complex so that such a code would perhaps be very lengthy, and a lengthy code would be self defeating as its length would prohibit it from serving its purpose — making clear to the user at a glance the functions that are being performed. Consequently in developing a hierarchic structure, careful attention must be given to the resulting code.

Since the computer operations are not all equal in the sense that mathematical operations are, a structured set of operations may prove to be unnecessarily confining. In mathematics and especially in arithmetic, one has operands which can be manipulated according to certain operations. For example, numbers can be added, multiplied or divided. Although these numbers may vary in magnitude, they are equal in the sense that the operations treat them in the same manner. No number is given preferential treatment (excluding the case of division by zero), and the operations themselves are comparable.

This comparability of operations no longer holds when computer operations are considered. To be sure, computer arithmetic instructions such as clear and add, add magnitude or subtract do have sir ar features. But what can one say about the relationship among the operations of loading, branching and I/O? It would appear that such instructions have little in common. In other words, computer operations are much "richer" than mathematical operations. Furthermore when micro-instructions are combined to form a machine instruction, it is usually found that the sequence alone is not sufficient to describe the instruction. Another parameter must be specified — time, or more precisely, the times between certain subsequences of micro-instructions. Thus, the inst ction is more complex by far than the sequence of micro-instructions which compose it.

Because of these reasons, the development of a structured set of operations which will encompass the entire computer appears not to be feasible. If one considers the problem combinatorially, it is evident that so many combinations of instructions (with their associated timings) are possible that an overall structure encompassing all such combinations seems not to be attainable. However, as in the case of the IBM 7094 codes, it may be possible to develop structured sets of micro-instructions which will apply only to specific classes of instructions such as control or floating point or shifting operations.

In conclusion, while much remains to be done in the development of a structured set of operations for the processor, it would appear that the concept is still a valid one especially for certain classes of operations.

TABLE XXXIX. COMPARISON OF PARTITION-TYPE CODES

I. Octal Codes for the IBM 7094 instruction set

00xx:	transfer and other control
02xx:	multiply, divide, floating multiply, floating divide
03xx:	floating, add, subtract, logical
04xx:	add, subtract, logical, control
05xx:	load, reset, control
06xx:	store
07xx:	place

II. Typical numeric codes for a business organization

00xx:	headquarters
01xx:	comptroller
02xx:	marketing
03xx:	engineering
04xx:	manufacturing
05xx:	distribution

TABLE XXXX. COMPARISON OF CHEMICAL AND COMPUTER HIERARCHIC STRUCTURES

	Chemistry	Computers
1.	electrons, protons	flip-flops, gates, levels
2.	atoms	micro-instructions for micro-programming
3.	elements Na, Ca, Cl, C	machine instructions CLA, TRA, ANA
4.	heavy hydrogen	special machine instructions, RDCA, RDCB
5.	chemical compounds $NaCl$, CCl_4, H_2SO_4	macro-instructions

NUMERIC PROCESSING

This section discusses how an associative memory may be used in problems that are numerically oriented. Areas in the field of numerical analysis are:

- Interpolation
- Numerical differentiation and integration
- Numerical solution of differential equations
- Least squares polynomial approximation
- Quadrature methods
- Approximation methods
- Numerical solution of polynomial equations
- Numerical solution of sets of linear equations
- Numerical methods for matrix inversion
- Numerical computation of the eigenvalues and eigenvectors of a matrix

It may be stated that for a computer to perform efficiently on these problems—whether or not it is equipped with an associative memory feature—the machine organization must have registers, adders, etc.

The question may be raised as to how an associative memory may be exploited to help solve numeric problems, or putting the matter more precisely: are there important numerical techniques which can be implemented most successfully on a computer with an associative memory? This question is nontrivial since many of the techniques which have been developed over the past several hundred years have been predicated on the assumption that computation would be done by hand. It is thus entirely possible that some of these techniques are not well suited to execution by a programmed calculator and indeed this turns out to be the case.

In order to obtain some results in depth, it was necessary to limit the scope of the inquiry to one or two important numerical problems. Because of their occurrence in a wide variety of applications, matrices appeared to offer a good choice for concentrated attention.

The chief numerical problems connected with matrix operations are the computation of the inverse of a matrix, the reduction of a matrix to either triangular or diagonal form and the computation of the eigenvalues and eigenvectors of a matrix.

In the course of computing solutions for one of the above problems, it often becomes evident that the results of the computation may prove to be meaningless because of the loss of precision due to round-off errors. It thus becomes clear that the magnitude of the matrix entries is of surpassing importance. If certain entries are much larger than others, then these numbers will influence heavily the entries in the inverse matrix as well as the size of the eigenvalues and eigenvectors. Consequently, many of the matrix reduction techniques developed over the years require for their success the identifying and relocating of the entries of largest magnitude.

The various pivoting techniques are more effective if the pivotal elements are the largest numbers respectively in each row. Now searching for a good pivot is feasible if the computing is being done by hand or with a desk calculator. However, such searching techniques clearly cannot be easily programmed for existing computers. They are too time consuming.

When an associative memory capability is added to a computer, however, the searching no longer appears to be such a problem. Perhaps the best way to

see how an associative memory can be used is to go through briefly the process of reducing an n x n square matrix whose entries are real numbers. The object is to reduce A to A' which is a matrix having non-zero entries along and above the main diagonal and zeroes below the main diagonal. Figure 22 depicts A and A'.

$$A = (a_{ij}) = \begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}$$

$$A' = (b_{ij}) = \begin{array}{cccc} b_{11} & b_{12} & \cdots & b_{1n} \\ 0 & b_{22} & \cdots & b_{2n} \\ \vdots & & b_{33} & \vdots \\ \vdots & & \ddots & b_{n-1,\,n} \\ 0 & \cdots & 0 & b_{nn} \end{array}$$

Figure 22. Reduction of a Matrix to Upper Triangular Form.

The reduction process will be explained as a series of steps.

Step 1

Search for the largest element in A and by interchanging rows and columns, bring this largest element to position (1,1). If this largest element had originally been in position (7,9), say, then after the interchange it will be at (1,1) and A_{11} will be at (7,9). Notice that the search operation is easily accomplished with an associative memory.

Step 2

Reduce the rest of the entries of column 1 to zero by multiplying row 1 by a_{k1}/a_{11} and subtracting from row k, for k = 2, 3, ..., n, respectively.

Step 3

Search for the largest element in the (n-1) x (n-1) minor (a_{ij}), where i = 2,...n, j = 2,...,n. Note that because of steps 1 and 2, these entries will be much different from the corresponding original entries in A. Bring this largest element to location (2,2) by interchange of rows and columns.

Step 4

Reduce the entries in column 2 which are <u>below</u> (2,2) to zero by multiplying row 2 by a_{k2}/a_{22} and subtracting from row k as k = 3,4,...,n, respectively.

Continuing this process, it is seen that in general after the entries of column k below (k,k) have been reduced to zero, the largest element in the remaining ((n-k) x (n-k)) minor is chosen and brought to position ((k+1), (k+1)) and the remaining entries below ((k+1), (k+1)) in column k+1 are reduced to zero.

Notice that the final computation will be that in which row n-1 is multiplied by $a_{n,n-1}/a_{n-1,\,n-1}$ and subtracted from the last row. The entry at (n,n-1)

w[illegible] be reduced to zero and the last entry at (n,n) will be simultaneously obtained. At that point, A will have been reduced to A'.

If this method is used to solve a system of equations $AX = T$, where X is a column vector of unknowns and T is a column vector of constants, then the interchanges called for by Steps 1 and 3 will also affect the components of X and T. To be specific, interchanging rows p and q of A will cause components t_p and t_q of T to be interchanged while interchanging columns U and v will result in the interchange of X_u and X_v. Certain bookkeeping routines will be needed to keep track of these changes. These will be explained in more detail later.

A few comments are in order regarding the role that an associative memory has played in this reduction. An associative memory has not improved or changed the manner in which the elements [illegible] the diagonal are reduced to zero. In other words, the strictly arithmetic processes called for proceed the same way in a computer with an associative memory as in a regular machine. Rather the chief contribution of an associative memory has been to facilitate the search for the largest element in the various arrays.

As mentioned above, the interchange of columns will cause the components of X to be switched. In order to keep track of these changes, it will be necessary to tag each vector position so as to indicate which of the original unknowns is now occupying which position in X. The following example will demonstrate the relabeling of the vectors.

Example. Suppose for definiteness that the matrix is 9x9. The unknowns then are denoted by the 9 component vector X_o where $X_o = (X_1^{(1)}, X_2^{(2)}, X_3^{(3)}, \ldots, X_9^{(9)})$. In explanation of $X_i^{(j)}$, the subscript i refers to the column

position in the array. The superscript (j) denotes which of the original vectors is occupying the i-th position. Naturally at the beginning i=j as is indicated by X_0.

Step 1

Assume that columns 1 and 9 are interchanged. Then X_0 becomes $X_1 = \left(x_1^{(9)}, x_2^{(2)}, ---, x_8^{(8)}, x_9^{(1)}\right)$.

Step 2

Clear column 1 below the diagonal.

Step 3

Assume that columns 2 and 9 of the new array are interchanged. $X_1 \to X_2 = \left(x_1^{(9)}, x_2^{(1)}, x_3^{(3)}, ---, x_8^{(8)}, x_9^{(2)}\right)$ Note that $x_2^{(1)}$ is correct because column 1 went to position 9 by Step 1 and then to position 2 by Step 3.

Step 4

Clear column 2 below the diagonal.

Step 5

Columns 3 and 9 switch.

$$X_2 \to X_3 = \left(x_1^{(9)}, x_2^{(1)}, x_3^{(2)}, x_4^{(4)}, ---, x_8^{(8)}, x_9^{(3)}\right) .$$

Step 6

Clear column 3 below the diagonal.

Step 7

Columns 4 and 8 switch.

$$X_3 \rightarrow X_4 = \left(X_1^{(9)}, X_2^{(1)}, X_3^{(2)}, X_4^{(8)}, X_5^{(5)}, X_6^{(6)}, X_7^{(7)}, X_8^{(4)}, X_9^{(3)} \right) .$$

Step 8

Clear ....

Step 9

Columns 5 and 7 switch.

$$X_4 \rightarrow X_5 = \left(X_1^{(9)}, X_2^{(1)}, X_3^{(2)}, X_4^{(8)}, X_5^{(7)}, X_6^{(6)}, X_7^{(5)}, X_8^{(4)}, X_9^{(3)} \right) .$$

Step 10

Clear ....

Step 11

Column 6 does not change. X_5 remains

$$X_5 = \left(X_1^{(9)}, X_2^{(1)}, X_4^{(8)}, X_5^{(7)}, X_6^{(6)}, X_7^{(5)}, X_8^{(4)}, X_9^{(3)} \right) .$$

Step 12

Clear ....

Step 13

Columns 7 and 8 switch.

$$X_5 \rightarrow X_6 = \left(X_1^{(9)}, X_2^{(1)}, X_3^{(2)}, X_4^{(8)}, X_5^{(7)}, X_6^{(6)}, X_7^{(4)}, X_8^{(5)}, X_9^{(3)} \right) .$$

Notice that $X_7^{(4)}$ and $X_8^{(5)}$ are correct because column 4 went to position 8 by step 7 and then to position 7 by step 13. Column 5 went to position 7 by step 9 and then to position 8 by step 13.

Step 14

Clear . . . . .

Step 15

Columns 8 and 9 switch.

$$X_6 \rightarrow X_7 = \left(X_1^{(9)}, X_2^{(1)}, X_3^{(2)}, X_4^{(8)}, X_5^{(7)}, X_6^{(6)}, X_7^{(4)}, X_8^{(3)}, X_9^{(5)} \right) .$$

Vector X_7 is the final arrangement and the original unknowns now appear in the order (9,1,2,8,7,6,4,3,5). Notice that in the overall process the original columns were changed as follows:

	Number of Changes
1→9→2	2
2→9→3	2
3→9→8	2
4→8→7	2
5→7→8→9	3
6	0
7→5	1
8→4	1
9→1	1
	14

It is interesting to observe that the maximal number of changes for a single column would occur if column 1 were successively interchanged with the next higher column at each step, resulting in the following arrangements as the final arrangement of the components: (2,3,4,5,6,7,8,9,1). In terms of changes to the original columns, this arrangement can be expressed as:

	Number of Changes
1→2→3→4→5→6→7→8→9	8
2→1	1
3→2	1
4→3	1

	Number of Changes
5→4	1
6→5	1
7→6	1
8→7	1
9→8	1
	16

It has been suggested in several previous tasks that certain tags should be associated with data items. From the foregoing discussion, it is evident that such tags would be extremely useful for these matrix manipulations, for it is clear that the column switching just described refers to a conceptual process, not to an actual one. That is to say, the nine components of the vector will not be physically moved about in the memory. Instead by use of the tag marker the components will be simultaneously relabeled, thereby effectively simulating the switching operations. The associative capability will thus facilitate the relabelling process by permitting parallel examination and relabeling of all the markers associated with vector components.

Turning now to the problem of computing the eigenvalues and eigenvectors of a matrix, it will be shown that an associative memory capability will significantly improve the computer implementation of one of the better known techniques presently in use for dealing with this problem.

The techniques referred to are the method of Jacobi which was developed for Hermitian matrices and which in practice is used chiefly for a real symmetric matrix. Reduced to diagonal form, the sum of the squares of the diagonal elements of the original matrix is always less than the sum of the

squares of the diagonal elements of the reduced form. For example the reduced form of

$$\begin{pmatrix} 7 & 6 \\ 6 & 2 \end{pmatrix} \text{ is } \begin{pmatrix} 11 & 0 \\ 0 & -2 \end{pmatrix} ; \; 7^2 + 2^2 < 11^2 + (-2)^2 .$$

Also, the reduced form of $\begin{pmatrix} 8 & 2 \\ 2 & 5 \end{pmatrix}$ is $\begin{pmatrix} 9 & 0 \\ 0 & 4 \end{pmatrix}$ and again $8^2 + 5^2 < 9^2 + 4^2$. The elements (11,-2) and (9,4) of the reduced forms are, of course, the eigenvalues of the original matrices.

The second fact underlying the Jacobi method comes from observing that if the above matrices are examined more closely it is seen that

$6^2 + 7^2 + 2^2 + 6^2 = 11^2 + (-2)^2$ and

$2^2 + 8^2 + 5^2 + 2^2 = 9^2 + 4^2$.

In other words, the increase in the diagonal elements of the reduced form over the diagonal elements of the original matrix is accounted for since the off-diagonal elements have decreased in magnitude to zero.

The method of Jacobi makes use of these facts by finding a set of transformations each of which decreases the off-diagonal elements and increases the diagonal elements. Performing these transformations in sequence on a real symmetric matrix A will reduce it to diagonal form.

The method requires that at each step, the off-diagonal element a_{ij} of largest magnitude be annihilated. Herein lies the problem of implementing this method. For computation by present machine organizations, the search for the largest element is too time-consuming. Consequently, the customary practice is to program the method so that the elements will be annihilated in

some pre-assigned cyclic order. Unfortunately the cyclic order—annihilation has two drawbacks. First, a larger number of transformations will be required. Secondly, and more serious it is possible that the sequence of transformations may not in fact yield the reduced diagonal form. That the Jacobi method does produce the diagonal form depends on annihilating the appropriate elements, i.e., the largest element, at each step of the process. Since this is not done when the elements are annihilated cyclically, special precautions must be taken to ensure that the cyclic process does converge to the diagonal form.

If on the other hand, the computer has an associative memory, the cyclic order annihilation process with its accompanying drawbacks can be avoided. Now with the associative memory capability it will be feasible and entirely practical to search for the largest off-diagonal element after each transformation. Consequently the Jacobi method can be followed strictly from which it necessarily follows that the eigenvalues will be obtained after performing the minimal number of transformations (as compared with the number required by the cyclic order process) in a sequence whose convergence is guaranteed.

As has been stated, a prime requirement in carrying out the Jacobi method—and one which is satisfied by the associative processor—is the ability to identify quickly the largest of a set of numbers. Now as the final diagonal form is approached, another, rather opposite, requirement appears. This requirement is the ability to determine when the off-diagonal elements have been sufficiently reduced in size so that they may all be considered to be, in effect, zero. Such a requirement will arise because round-off and truncation

errors make it likely that the off-diagonal elements will never vanish exactly. Evidently, the associative processor will satisfy this requirement for it is as equally capable of determining the smallest as well as the largest of a set of numbers.

Summarizing then, one may conclude that the associative processor is uniquely able to handle problems that require identification of the largest (or smallest) of various arrays of numbers. Moreover this capability becomes increasingly valuable when, as is the case here, the arrays are not specified in advance but instead are formed dynamically during some mathematical process.

Section VIII

GENERAL PURPOSE ASSOCIATIVE PROCESSOR

INTRODUCTION

The major design effort was concentrated on finding hardware configurations for solving problems which were identified in the initial phase of the project. Studies on individual problem types resulted in designs which have been described in other sections of this report. The remaining design effort, described in this section, was to integrate selected logic of the individual problem processors into an overall machine design. In addition, another study was performed on the possibility of applying mathematical theories of structure to determine a set of operations for the processor, as reported in Section VII. The design of the "overall processor" had the primary goal of exhibiting improved performance in non-numeric areas over conventional general purpose computers of the sequential von Neumann type. In particular, increased speed and ease of programming were sought for non-numerical data processing problems and the processing of problems involving symbol manipulation.

DESIGN INTEGRATION GOALS AND APPROACH

The design of the processor was approached with a view toward general purpose computer capability, integrated with logic for processing non-numerical problems, of the types studied, to retain the advantages of associative processing on non-numerical data. This may be expressed as a "covering problem," as

follows: given a set of individual processor designs, find a machine organization in which a set of data paths and machine operations covers the individual processor designs. For perfect "covering," the machine should be indistinguishable from each individual processor in performing the task for which the processor was designed. The problem of "covering" is not precisely defined, although its qualitative nature is clear. The problem is somewhat similar to the design of an efficient program, organized to use common subroutines as much as possible. This analogy suggests a potential loss of efficiency, analogous to loss of execution speed due to linking operations for utilizing subroutines. A key issue is economy: an economical solution to the covering problem is desired so certain sub-machine organizations (memory, register data paths, and machine operations) are common to many processor problems. Obviously the design problem is not one which has a unique solution, nor is it one for which the objective criteria are at hand for judging a solution.

An approach, which is more of a guide than an objective measure of design, is to seek performance according to the expected usage of the machine. Assuming numeric processing will utilize machine facilities only 10% to 20% of the operating time principal emphasis should be on features for non-numeric processing. Related to this aspect is the question of how well a proposed design for the processor satisfies the requirements imposed by the need to "cover" the individual problem processors, with due weighting attached to efficiency of execution according to the percentage of machine operation time which will be used on each of the problem types.

The present trend toward making a machine more usable to a community of users (the community, as meant here, is comprised of technical or military specialists of an organization) will be an established "modus operandi" by the era of the general purpose associative processor (GPAP). Thus, hands-on programming, time-sharing, real-time computing, and monitor operation are required capabilities. The total design of such an advanced operating system is beyond the scope of this effort, but hardware for the system will include the required features. An advanced system capable of operating under supervisory control is intended. The operational system will have the capability for building on concepts which are foremost in the current state-of-the-art, whose major innovations are related to the incorporation and usage of associative memories and/or devices in the system.

A goal for the advanced processor is ease of programming. Particular properties of the associative processor which will help achieve this goal are inherent in the ability to designate data by its name or by a property of the data. Man can supply this information easier than he can give the address of data in memory. For problems too difficult to think through to the point of determining the complete computer program and coding such a program for a machine, computer languages are needed. These languages must be oriented to the way people think through their problems; man-machine interaction is needed. The use of computer languages is simplified when the human specifies "goals" or some description of the result of computation, rather than the detailed steps by which the computer achieves the result.

Another aspect, possibly the most challenging task for advanced computer design, is the discovery of techniques for efficient and effective man-machine communication. This goal is only partially achieved by making the machine an interpreter of the language which the user finds most convenient for his problems. Additional features for man-machine communication would consider the operational environment of the system in more detail which is outside the scope of this study. This study did consider the need to allow a programmer to either use the language best suited for his application, or design innovations and modify the language without concern for developing and debugging software packages to compile his language.

THE GPAP SYSTEM

The machine organization for the associative processor is presented in Figure 23. A relatively conventional general purpose unit (GPU) is shown, which executes instructions obtained from its core memory. An instruction which indicates that an operation is to be performed by an associative unit (AU) will also identify the AU, and cause the GPU to make tests to determine whether the AU can perform the operation. If successful, operation will be initiated by transfer of data from the GPU to the AU to define the operation to be performed. Also, instructions will be given by the GPU to the Control Unit, the Multiplexor, the AU, and to the device (which may be the GPU, itself) which will supply the data on which the AU will operate.

Inputs and Outputs of the Associative Processor may be transmitted to and from terminals and devices through I/O Channels, which are also interconnected

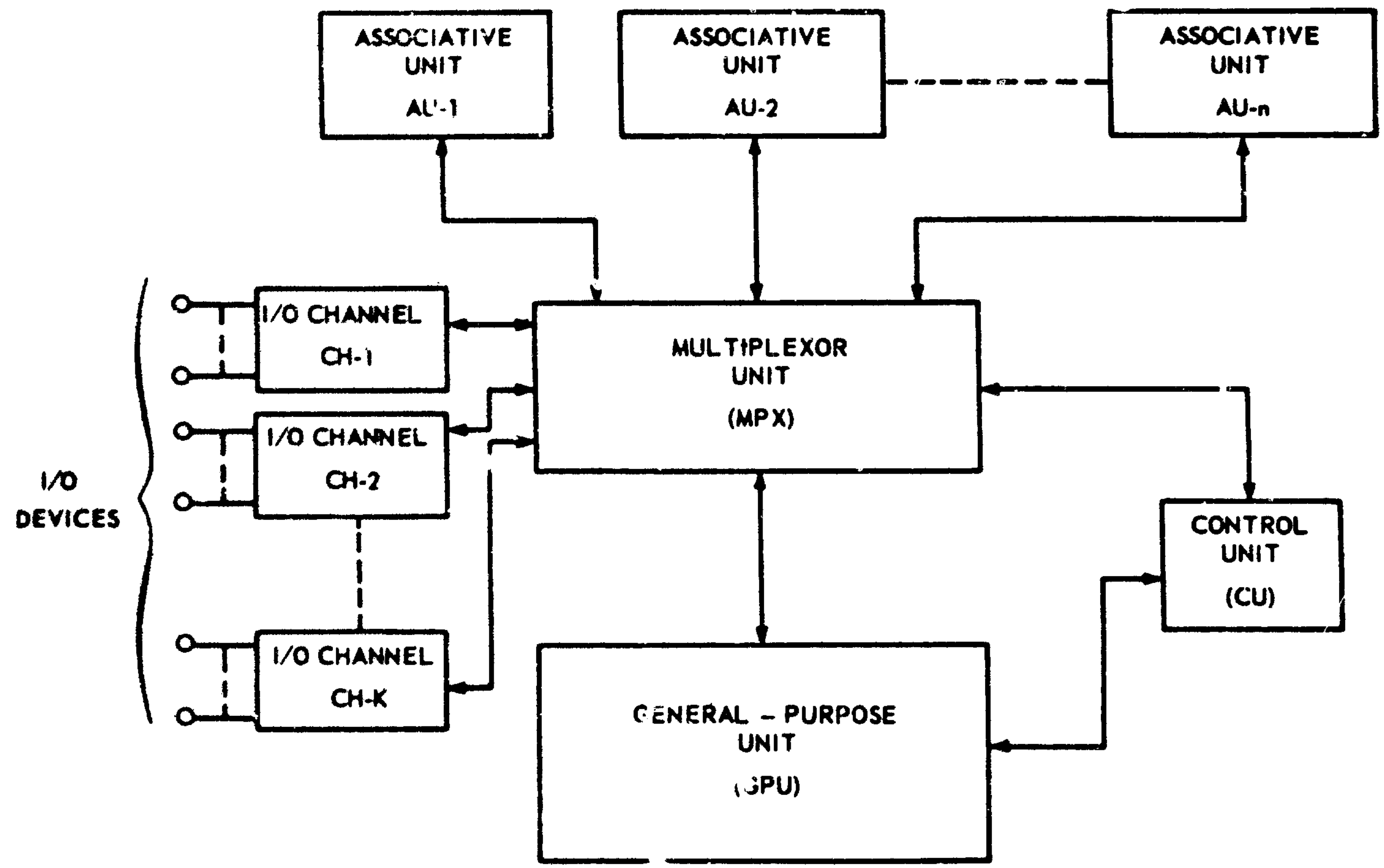


Figure 23. Associative Processor Machine Organization

to the GPU and the AU's by means of paths established by the Control Unit through the Multiplexor.

Each Associative Unit contains an associative memory, registers, control, clocks, and a Microprogram Sequence Memory (MSM). The MSM is the means for specifying functions to be performed in the AU. At any one time, a MSM contains several microprogram sequences which it can execute on data in its associative memory. A MSM sequence may be an algorithm for a complex function which can be done more efficiently using associative memory rather than random-access core memory; for example it may be a text searching or inner product algorithm. The MSM program makes the AU perform in a manner similar to that described for a special purpose associative processor on the particular problem type.

All communication within the system takes place through the multiplexor, and a direct path may be established between any two elements. This feature eliminates the bottleneck which could occur if all data transfers were required to pass through the memory of the GPU. However, central control resides in the program which is executed by the GPU. Associative Units, the Multiplexor, and the Control Unit will respond and initiate action only when commanded by the GPU. Status information is transmitted by all functional units to the GPU, for its use in executing instructions in the GPU program.

The General Purpose Unit, (GPU), is an advanced design general purpose computer, current with the state-of-the-art. Its precise specification depends on the number of associative units and I/O units attached and the amount of activity for the system. Since the burden of supervising control to be exercised

by the GPU is very large, the GPU will have features to optimize its capability for fast reaction to external events. An associative memory is used to store the system's status, whereby the GPU can instantaneously determine what AU's, if any, are inactive and can be assigned to a task to be performed. When a new task is to be performed, its priority can be used to determine if any AU has a lower priority task and can be preempted to service the new task.

It is assumed that the GPU, MPX, and the AU's are located in close proximity, perhaps even in the same equipment housing, so that memory to memory transfers may be accomplished at high speed. Transfers between the GPU and the associative memory in an AU take place over the same memory bus by which the GPU accesses its own high-speed random-access memory. The Control Unit uses signal paths to establish a state of "readiness to operate" in units which are to be functionally interconnected. When a channel or an AU has correctly responded to control instructions, the Control Unit se the data paths for instructions and/or data to be transferred from the GPU to the Associative Unit. Initial instructions specify the task to be done and parameters of the task. Data on which to operate may come from the GPU or directly from one of the attached I/O devices, via I/O channels, to the Associative Unit without going through the GPU. The termination of a task, or the production of a result in the AU which should be communicated to the GPU, will be accomplished by sending the AU's signal for attention to the GPU. The normal point for a task to be completed will be predetermined by the GPU, as an address in MSM which will be reached for task termination. The Interrupt Control will operate when the termination address in reached in the MSM AR. This generates a signal to be transmitted to

the GPU. The GPU monitor program will control responses to the requests for attention by the Associative Units.

One (or more) of the I/O channels may be the associative channel processor as described in Section V. The response to queries may be transmitted directly from the disc file processor, to the associative memory of an AU, where the data can be processed by a microprogram set up with parameters by the GPU program to achieve a desired result.

THE ASSOCIATIVE UNIT

For generality, the Associative Unit (AU) has features which permit it to be programmed to perform the non-numeric problems which were described in other sections of this report. The general organizational structure is shown in Figure 24. Input and output take place through a buffer register, which is coupled to the memory data bus of the GPU for high speed data transfer, or to one of the I/O channels via the multiplexor for input and output to terminal devices. The GPU sends an initial command to specify what unit of the AU will be involved in subsequent data transfer, and how many words of information will be transferred. This command is decoded in the AU control circuitry and gates are set to make the proper data paths operative, in and out of the buffer register. In order to assign a task to the associative processor, the GPU transmits commands to the AU. The commands are macros which contain data for sequencing and parameters of the micro routines stored in MSM, and thus sequence the microroutines into a program which performs the operations. A start signal is given for retrieving the first micro instruction of the task. At the conclusion

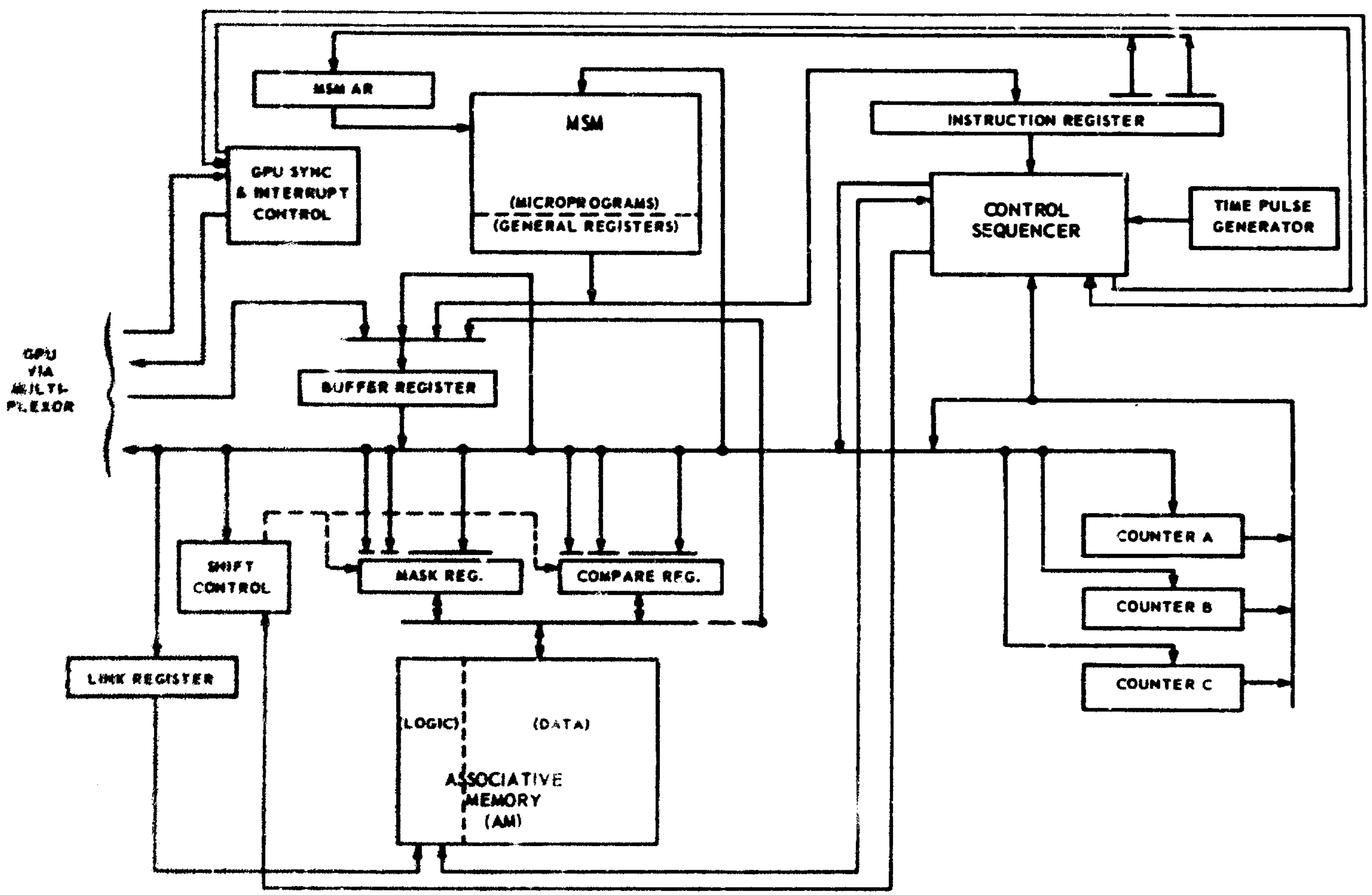


Figure 24. Organization of Associative Unit

of a task, the Associative Unit signals the GPU and awaits further instructions on what to do with the processed data, or operations to perform.

Once a task is started, the AU operates autonomously, taking microinstructions from its MSM, interpreting them through the control logic and executing them. The sequencing of microinstructions has been provided within the microroutines, and between microroutines by the macroinstructions given by the GPU. Coding in the microinstruction specify alternatives in sequencing when conditions in processing cause certain condition codes to be indicated.

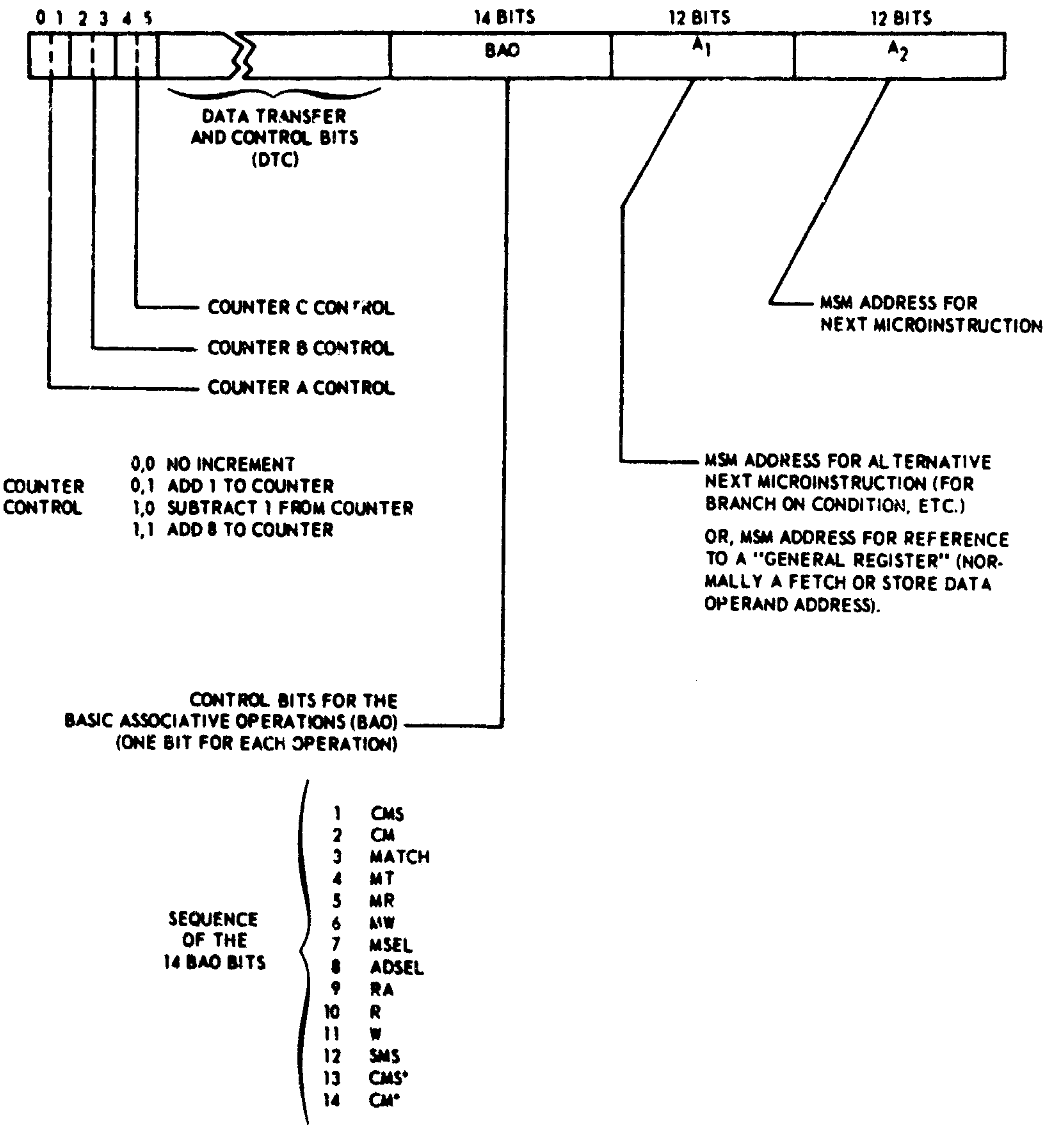
Counters are provided for controlling shift operations of the Mask and Compare Registers, as used (for example) in locating a pattern which may start on any bit or character position within an associative memory word. Once the starting position of a search pattern has been found, the same shift will be exercised in subsequent word searches, when the search pattern is longer than one word in the associative memory. Hence, the shift which was made in finding the start of the pattern match is needed for subsequent matches. The counter contents which specify shifts may be used as part of the compare data, and also stored in memory as "tag data" which is to be matched in succeeding interrogations for determining sequences of words in a search. The need for this feature arises from the logic of text processing and pattern classification, where the data in the compare register may match data words in the memory, but they are not initially in exact alignment at memory word boundaries.

The counters, A, B, and C, each contain 12 bits, and are incremented by time pulses from the control circuits. The counters are general purpose and may be used to control shifting of the mask and compare registers of the

associative memory, or to set the address register of the MSM or the AM. It may be noted that the address register and link bits of the AM are shown as a portion of the compare register in Figure 26. They are loaded and unloaded separately from the 72 data bits, and are not involved in shift operations on the Compare Register. Through control of loading the counters, incrementing them, and being able to specify their data transfer destination, microinstructions are the means by which the programmer determines the function of each counter, as explained with reference to Figure 25.

Data Transfer Control (DTC) is used to control designated transfers of data between registers. One bit of DTC is interpreted as meaning that the three counters are to be loaded with 36 bits (12 bits for each counter) which are obtained from the MSM register whose address is given by the A_1 field of the microinstruction. This is, however, but one illustration of the interpretation of a microinstruction.

Another bit of the DTC, if set to "1", means that all designated data transfers are to take place, including fetching an operand from a general register in MSM (if ordered) and the sequence of designated Basis Associative Operations (BAO) is to be executed. The DTC bits include assignment of counters to functions; for example, the "C" counter may be assigned to supply its contents to the shift control for the mask and compare registers. In the same microinstruction, the first six bits indicate what incrementing should take place in each counter prior to execution of operations, as shown in the format of Figure 25. The functions of DTC bits are directly concerned with data transfers whose detailed description would involve a level of detail not practicable at the conceptual stage of



*Note that these operations are repeated in the sequence so that they can be done at the end of a sequence of other operations.

Figure 25. Format of Microinstruction for Associative Unit

design. General functions only can be indicated, while specifying data path width, location of gating circuits, and control circuit conditions would involve fairly conventional design practices.

It is implicit in the description of the Basic Associative Operations that the Link Bit Register could be loaded by either DTC bits or by 6 bits from the contents of a MSM register. The design decision favors loading the Link Bit Register by latter method, i.e., with 6 bits from a MSM register, which may usually be done at the same time as the "load counter" operation, described above. One reason for this choice is that the data content of the Link Bit Register is not likely to change within a microroutine. This discussion serves to illustrate the concepts and functions involved in completely specifying the DTC bits. It is evident that details are best resolved when a detailed design is produced for implementation by hardware. No difficulties are expected in applying the state-of-the-art design techniques to cover the functions of DTC bits. Further description will, therefore, emphasize the associative operations and the logic connected with the associative memory.

Microinstructions are fetched from the MSM, to the Buffer Register, then transferred to the Instruction Register which supplies, signals to the Control Sequencer (CS). The Control Sequencer is also supplied with time pulses by the TPG and the clock. The logic of the CS is to gate pulses which establish data transfer paths and signals to execute operations on data. The most important of these for the discussion of the AU are the basic associative operations, shown as the BAO field of the microinstruction. The BAO bits are sampled in sequence and transmitted to the Associative Memory (AM) as signals to cause operations

in the memory. Since different time intervals are required to perform different basic operations, and a given operation may have variable time required for its completion, the jump from one basic operation to the next generally depends on receipt of a response from AM, indicating completion of the last operation sent to it. Further explanation of the basic associative operations will be deferred until a more detailed explanation of the memory and its logic are presented.

Microprogram Sequence Memory

The program for the associative processor resides in the Microprogram Sequence Memory, which is initially loaded by block transfer of a program from the GPU. The program is a sequence of microprogram instructions, produced by a suitable higher level language compiler (or interpreter) in the GPU. Thus it is presumed that a suitable program language will have been developed, and its compiler will exist in the GPU for producing the detailed microprogram sequences which actually control the operation of the associative memory.

Each step of a microprogram sequence consists of one or more basic functions to be performed during one Microprogram Storage Memory (MSM) cycle. Complex operations, such as "match greatest," will require a sequence of MSM cycles, and still more complex algorithms, such as ordered retrieval, will require sequences of complex operations. The MSM is used functionally as a read-only memory in the Associative Unit operations, but it can be written by the GPU. Thus, the assignment of a task, or special-purpose function by the GPU must be preceded by transmittal of the appropriate MSM sequences, to guide the further processing operations of the Associative Unit.

The MSM is a "conventional" random access memory which stores microprogram words, or instructions. Its contents may be completely changed by reloading it from the GPU. The advantage of a random access memory for the MSM is in the capability to alter the processing function of the AU by compiling a new microprogram sequence in the GPU; specialization of design is then accomplished by software and by establishing a library of microprogram sequences through which many specialized designs may be accommodated in one Associative Unit. The greatest advantage of a MSM with read/write capability will be realized in the early stages of GPAP usage, when the detailed design of specialized functions is still in the development stage. Thus, the choice of a conventional random-access storage over a read-only storage seems to be justified.

The capacity (number of words) of the MSM is determined by considering that it is desirable to minimize the amount of communication of the AU with the GPU. The cost of random-access core storage generally favors increased core storage as opposed to additional logic and the loss in effective computing time in both the AU and the GPU to deal with interruptions of processes. The capacity of the MSM will provide for storage of a set of sequences which could include all functions necessary for the text processing problem. The width of the MSM is determined by considering the number of basic functions for which control signals may be developed during one MSM cycle, and the requirements for sequencing, branching, testing conditions, etc.

Physically the MSM may be considered to have improved characteristics compared with present day fast scratchpad memories, with an order of magnitude greater capacity. Thus, a read/write cycle time of 200 nanoseconds or

less and an access time (for read only) of about 100 nanoseconds will be assumed. A capacity of at least 2048 words, of 36 bits each, or 1024 words of 72 bits each will be required. It is expected that magnetic core technology will still have economic advantages in memories which do not require a significant combination of the logic and storage functions; hence, while integrated circuit technology is postulated for the associative memory, magnetic technology is postulated for the MSM if the timing requirements can be met.

Another important use of the MSM is the program controlled use of MSM for general registers; i.e., registers for storing intermediate and final results of processing, or data and constants which will be needed for problem execution. Some consideration was given to including more registers in the Associative Unit; one might desire two or more sets of Mask-Compare Register pairs for complex searches. For example, the contents of one Mask/Compare might be stored in all Associative Unit words which match another Mask/Compare pair. By the ability to designate certain MSM registers as pseudo Mask/Compare pairs, and transfer these in and out of the physical Mask/Compare registers, the same capability can be obtained, but at a sacrifice in speed.

Associative Memory

Central to considerations in the design of the associative processor are the basic associative operations which are to be performed on data in the memory. This philosophy emphasizes that once the basic associative operations and the memory, itself, are specified, the necessary control signals for memory operations are revealed. Then the registers, data paths, and control signal generation

can be specified as necessary to support translating a machine instruction set into the proper operations on the associative memory. The basic associative operation set was derived from the earlier studies and individual machine designs for selected non-numeric problems; the text processor being especially significant in this regard. Conclusions were reached about the structure of the associative memory and its basic associative operations.

1. The prime associative operation is, of course, a matching of the contents of all words in the associative memory against some configuration of binary variables in an associated data register. In addition, it is desired to be able to indicate linking between contiguous words which are matched on successive matching operations against a string of data.

2. While stored data may be accessed associatively, extrinsic (or location addressing) means are also desired. The need to locate data by its storage location, especially by its distance from some other datum, arises in text processing. Also, extrinsic addressing simplifies other functions, such as input/output.

3. It is desired to be able to read or write any number of selected bits in words which meet the match test described in 1. Furthermore, the read or write operation should be performed on a single word, if desired, or simultaneously on all words which match the criteria specified by a data register and a mask register.

4. The evaluation of the result of a matching operation is required to determine if there was one matching word, more than one, or exactly how many words in the associative memory met the matching criterion. Since the match resolution consumes time, this is to be a program controlled operation and is not performed automatically after each match operation.

5. A sufficient (but small) set of basic associative operations is required. Complex functions may be performed by designing logic within each word cell of the memory, but such complexity tends to over specialize the memory and additional components required quickly reach prohibitive proportions. More complex operations than those indicated in 1 - 5, will be done by algorithms employing the basic associative operations.

6. The associative memory capacity is, nominally, taken as 4096 words with 72 data bits each. In addition, each word shall have cells which store the match status, the "match select" status, and the linking status by which connectivity or chaining can be determined.

The Associative Memory is shown in Figure 26. For the sake of discussion, the memory is assumed to be 4096 words, with storage capacity of 72 bits for each word. A word may contain 72 data bits, or it may contain fewer data bits if some bits of each word are required for tagging memory registers. The functions of tag bits, their location and the number of them in each word is a program controlled function; a dashed line is shown in the storage to indicate that the division of storage into data and tags is a flexible boundary. The memory is divided into two functional parts: Storage and Word Operation Logic. For each register (or word) in memory, there are directly associated components in each part which belong to the register. The functional division may also be a physical division, since a different technology may be chosen to implement each part, with, perhaps, economic advantages in doing so.

In view of the many efforts and promises to develop cheaper, faster memories by plated wire, ferrite slab, thin-film, and other magnetic technologies,

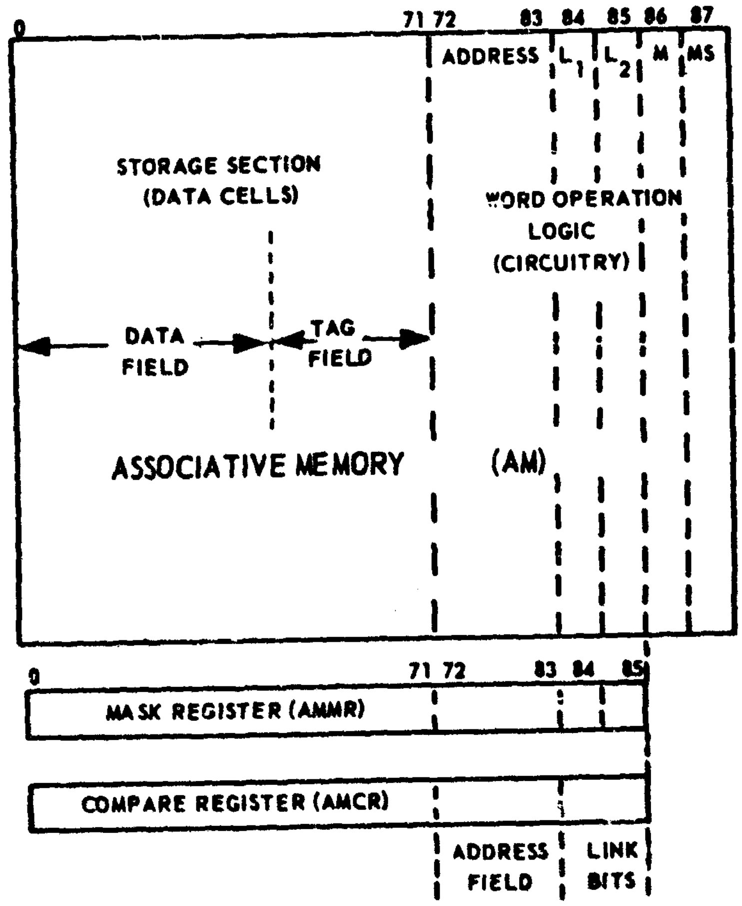


Figure 26. The Associative Memory

one of these may be selected to implement the storage part of the associative memory. It is assumed that the expectations for monolithic integrated circuits will make them economically feasible for the circuitry of the Word Operation Logic, certainly by the 1970 era. In fact, it may be expected that the entire associative memory may be comprised of monolithic integrated circuits in a not-too-advanced time era, if some predictions for this technology are realized. Accordingly, the functional division of the Associative Memory is, at least, a division into a part which is data cells and a part which contains circuitry and logic by which operations are performed in the memory.

The concept of a mixture of technologies in implementing the memory can be extended here to include the use of a read-only type storage for the address part of each word. Since the address will not be required to change during operation, active cells are not required. A caution in exploiting the use of mixed technologies is the usual problems of incompatabilities of electrical signals required by devices; this matter has not been investigated during the present study and comments regarding mixed technologies are, therefore, advanced as ideas to consider for future study.

While storage of the register address, i.e., the conventional sequential or random-access address, is shown in the circuitry, it could be omitted and 12 bits of the data portion could be used for addressing registers in those applications which require extrinsic addressing. The functions which involve the address would then be done by algorithms, instead of by circuitry as described herein. However, it was felt that the advantages of hardware for addressing are sufficient to warrant its inclusion, as we will discuss in describing the functions performed with Word Operation Logic.

The Mask and Compare registers are used in what is now a "conventional" way for associative memories. If the Mask bit is a "1" the corresponding bit of the Compare Register is used to determine whether a match on "1" or "0" is to be tested for the corresponding data bit of every word in the associative memory. It will be noted that the Mask and Compare registers overlap the address field and bits L_1 and L_2 of the Word Operation Logic. By overlapping the address field, word selection is enabled since the Match Bit (M) will be set for only that word in storage whose Address Field matches bits 72 through 83 of the Compare Register when the corresponding bits of the Mask Register are all set to "1". By setting a "1" in bit 72 of both the Compare Register and the Mask Register, and all "0's" in the bits 73 through 83 of the Mask, one half of the memory words will be set to "Match" in a Compare operation; similarly other "areas" of memory can be selected and identified, if it is desiraible to do so, by using partial addresses. Bits 84 and 85 of the Mask and Compare Registers are "link bits", whose operation will be evident in the description of the basic associative operations, next.

BASIC ASSOCIATIVE OPERATIONS

Each basic associative operation may be programmed as a bit in a micro-instruction, and the operations are executed sequentially during execution of each microinstruction. This section describes the basic associative operations to provide information needed by a programmer if the machine were to be built. Reference should be made to Figure 28 in reading this section. Also, the

definition of "word state" is needed. The word state is expressed by the conditions of two flip-flops, MATCH and MATCH SELECT, for each word of the memory. Basic associative operations are effected by control signals from the Control Sequencer of the Associative Unit, but the operation is performed on a given word only if its state is in the required condition.

1. Clear Match Select (CMS) If the MATCH SELECT flip flop is "on" it will be turned off: i.e., if the word state is either (0, 1) or (1, 1) then operation takes place to make the resultant word state (0, 0) or (1, 0), respectively.

2. Clear Match (CM) For all words of the memory, if the word state is (1, 0) the MATCH flip flop will be cleared and the resultant word state is (0, 0). Note that a word whose state is (1, 1) will not be affected; this provision allows programmers to operate on words and leave them in the (1, 1) state if they wish to eliminate such words from being processed by succeeding operations.

3. Match (MATCH) The MATCH operation sets the MATCH flip flop of all memory words in the (0, 0) state, so that they go to the (1, 0) state. An interrogate signal is generated which samples gates on each data bit of the words which make the transition (but not on words which were previously in the (1, 0) state). The interrogate signal will be gated as a NO MATCH signal for any data bit which does not agree with the unmasked data in the 74 bits of the Compare Register, of which 72 bits are the data portion and 2 bits are the link bits, L_1 and L_2. The result of a NO MATCH signal from the gated interrogation signal is that the word state reverts to (0, 0); otherwise, the word state remains as (1, 0). In a word which was not reset to (0, 0), the delayed interrogate signal sets bits L_1 and L_2 of that register, the preceeding register, and the following

register according to the oontents of the Link Register. Recall that the Link Register was specified as 6 bits. The first two bits control L_1 and L_2 for the preceeding register; the second two bits for the matched register; the third two bits for the next register. Furthermore, control of Link bits is "set" only, not reset, and the setting of a link bit is an OR of the conditions which can set it. For example: if the contents of the Link Register are 100110, then the L_1 bit in the preceeding register to each match register is set, the L_2 bit in each matched register is set and the L_1 bit in each following register is set. If the match field contains ABC and storage is as below, then the storage will be as indicated after the match operation for the above link register configuration.

Match Field/	L_1-bit/	L_2-bit/	M-bit
ABX	0	0	0
ABY	1	0	0
ABC	0	1	1
ABD	1	1	0
ABC	1	1	1
ABC	0	1	1
AXY	0	1	0

4. Match Test (MT) A "match test" signal is emitted (from the Control Sequencer) which polls the state of words in the memory until one is found in the (1, 0) state, then the signal is gated through the word logic and returned as a "match" signal to set a "match status" flip flop in the Control Sequencer. The "match status" is one of the conditions on which the current (or a successor) microinstruction can be programmed to select an alternative address for the next microinstruction. If no word in memory is in the (1, 0) state, the match status flip flop is reset by a signal from the last memory word to be polled. No word states are changed by the MT operation.

5. Multi Read (MR) The contents of all memory words in the (1, 0) state are written (as a Boolean OR) into unmasked bit positions of the Compare Register. No.word states are changed.

6. Multi Write (MW) The contents of unmasked bit positions of the Compare Register are written into corresponding bit positions of all memory words in the (1, 0) state. No word states are changed.

7. Match Select (MSEL) A "first-match test" signal, emitted from the Control Sequencer, polls the state of memory words in succession, starting at the lowest address, until a word in the (1, 0) state is found. The first word in the (1, 0) state is changed to the (0, 1) state and returns a "first match set" signal to the Control Sequencer. If no word is found in the (1, 0) state, a "match status reset" signal is returned to the Control Sequencer.

8. Address Select (ADSEL) The memory word whose extrinsic address is the unmasked portion of the Address Field of the Compare Register will be set to the (0, 1) state, no matter what state it had been in.

9. Read Address (RA) The extrinsic address of a memory word in the (0, 1) state is transferred to the Adddress Field of the Compare Register. If more than one memory word is in the (0, 1) state, the contents of the Address Field will be a logical OR of the binary representation of the addresses of those words. The word state is changed from (0, 1) to (1, 1) by this operation.

10. Read (R) The unmasked bits of the Compare Register are set to match corresponding bits in a memory word which is in the (0, 1) state. If more than one memory word is in the (0, 1) state, the read-out results in an OR of the contents of those words into the Compare Register, giving a "selective multiread"

capability, but the expected usage of the read operation is to transfer the contents of one memory word, only, to the Compare Register. The word state is changed from (0, 1) to (1, 1) by this operation.

11. Write (W) The unmasked bits of the Compare Register are written in any memory register which is in the (0, 1) state. The word state is changed from (0, 1) to (1, 1) by this operation.

12. Set Match Select (SMS) Any memory word in the (0, 1) state, as a result of a previous MATCH, is put in the (1, 1) state. This provides a useful means for selecting registers (by MATCH) and putting them into an inactive status, i.e,, into the (1, 1) state, from which they can subsequently be recalled by a CMS operation. The word state is changed from (0, 1) to (1, 1) by the SMS operation.

Word Operation Logic

The associative memory and electronics associated with its operation are indicated in Figure 27, with detail for only enough of the logic to indicate what is involved in basic associative operations. Assume that some data configuration exists in the Associative Memory Compare Register (AMCR), and in the Associative Memory Mask Register (AMMR). An interrogate signal, simultaneously applied to all words in the memory will test each data word to determine if its data matches the unmasked data in AMCR. A signal will be produced for any word in which one or more bits fail to match. The No Match signal is input to the Word Operation Logic, which is designed to control operations on the word

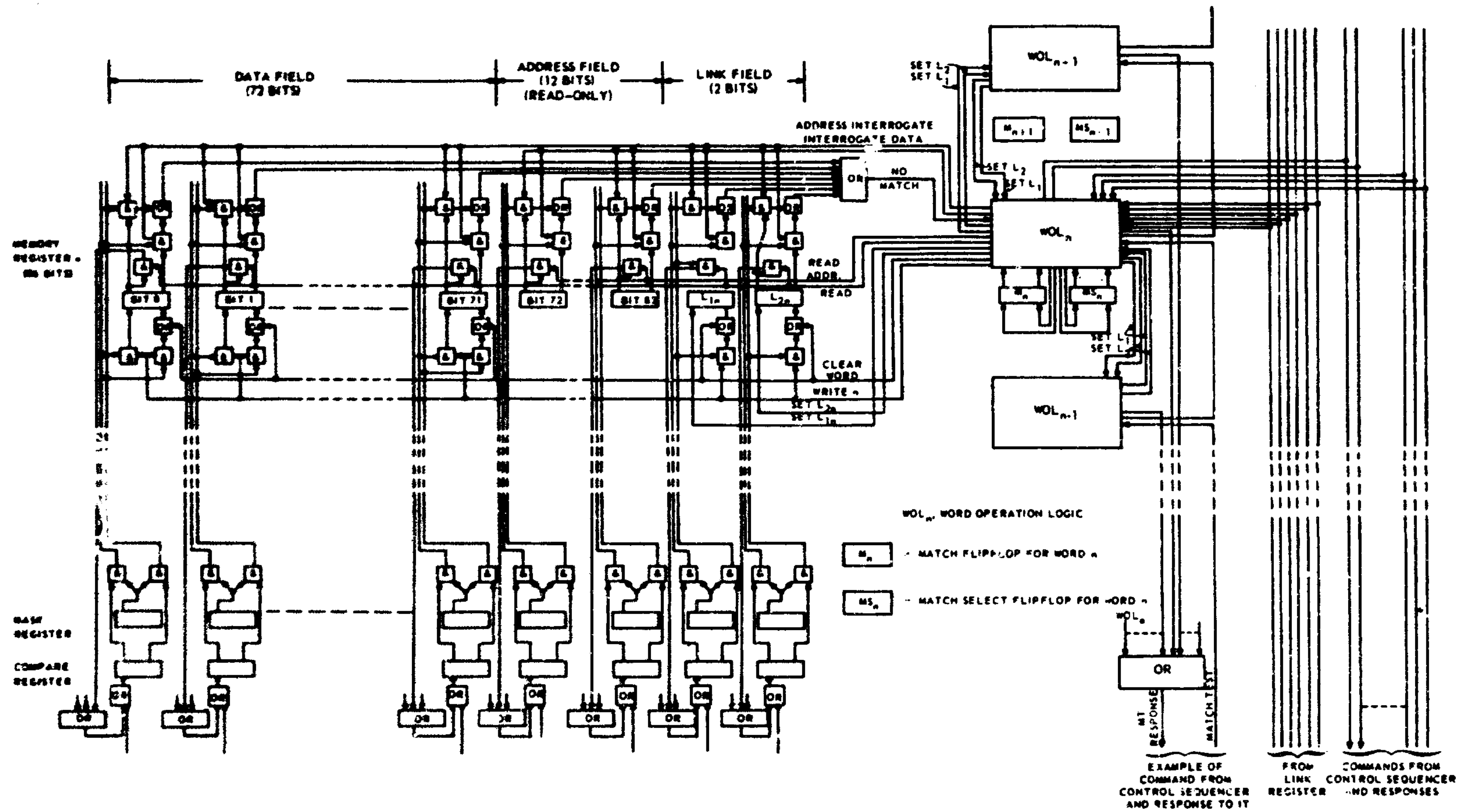


Figure 27. Electronics Associated With a Memory Register

with which it is associated. The Word Operation Logic performs functions to coordinate and respond to processor control signals. Further explanation of the Word Operation Logic will clarify the role of the basic associative operations and make their meaning more precise, but before proceeding further the reader is reminded that the design intent was to leave open the selection of specific technology, at least for the data storage part of the associative memory. The logic as shown here is intended to indicate logical functions, not the specific implementations of them, which would, in implementation, take best advantage of characteristics of the chosen technology.

The Word Operation Logic (WOL) exists for each word in associative memory. One set of inputs to WOL is from two flip flop cells in the associated word, which are called Match and Match Select. Match indicates whether the contents of the word agree with the unmasked data register contents in a prior interrogation of the memory. Match Select indicates whether the word has been selected for an operation successive to the matching process. Other inputs to WOL are the No Match signal, emitted when the AM word fails to match the masked data register, and control signals from the processor which depend upon the microprogram being executed. The control signals, in combination with the logical state of the word, denoted as (M, MS), determine the next state of the word. To further explain the response to control signals, consider Figure 28, in which the word states are shown as (0,0), (0,1), (1,0) and (1,1) to indicate the states of M and MS. Transition from one state to another is made on occurrence of a command signal, as indicated.

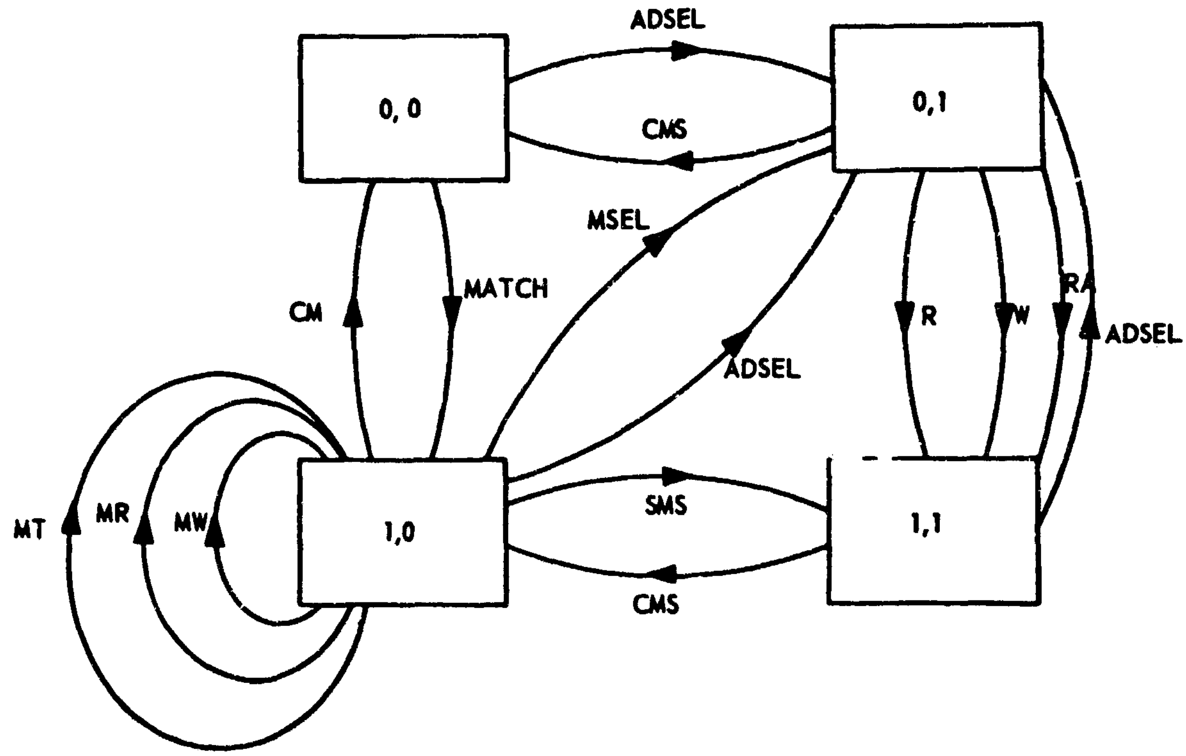


Figure 28. Associative Memory Word State Transition Diagram

The Word Operation Logic also sets the "link bits", L_1 and L_2, of each word, when appropriate conditions are met. As a result of performing memory interrogation on a MATCH command, L_1 and L_2 are set according to the contents of the Link Bit Register. Thus, L_1 and L_2 may be set by either of the words continguous to the word in which they belong.

The state (0, 1) is unique in the respect that one, and only one, word of the memory may be put in the (0, 1) state at one time. A Match Select (MSEL) operation must procede a read address operation, or a read or write operation which is to be performed on one word only. Note that an address select command always sets the state to (0, 1) for the selected word, no matter what state it was in prior to the address select. All conditions which prepare the memory so that a unique selected word is to be operated on must leave one and only one word in the (0, 1) state.

The microprogrammer provides the proper command sequence. For example, the sequence

(1) Clear Match Select (CMS)

(2) Clear Match (CM)

will always put the words in the (0, 0) state; in fact these two operations can be done at one time by Clear Match and Clear Match Select bits in a single microinstruction. The operation following this may be an Address Select (ADSEL) or a Match, Match Select (MSEL) sequence to obtain one word of the memory in the 0, 1 state.

Multi-Read (MR) and Multi-Write (MW) capability can be properly used only when the words to be operative in the read or write are in the (1, 0) state,

and all other words are in some state other than (1, 0). If all (or even some) matched words have been operated upon by operations on a word-by word basis Read, Write, or Read Address) and it is desired to start a new operation, it is unnecessary to Match again if the match criteria have not changed, since the CMS signal will restore all words in the (1, 1) state to the (1, 0) state.

Note that Set Match Select (SMS) changes the state from (1, 0) to (1, 1). This operation may be used to initially select a set of words (by Match) and then temporarily exclude them from any other operation. SMS will not be operative on any word not in the 1, 0 state. The usefulness for set theoretic operations can be illustrated by considering that all words of memory can be matched on characteristics A and B; then a IS command is given to eliminate all words in the conjunction of A and B. Fol wing this, separate matches on A and B will give the sets $A\overline{B}$ and $\overline{A}B$, respectively. If desired, the membership in each of the three sets AB, $A\overline{B}$, and $\overline{A}B$, could have been tested by algorithmic counting; if these three sets are included in a large set whose membership is also required, it, too, can be identified by MATCH operation and a following sequence of MSEL and counting operations. This sequence of operations is suggested, for example, in pattern classification programs.

Another capability for setting aside words which are not to be considered in a successor operation is through the use of the tag bits which the programmer may define in the data part of the memory. The procedure would be to first identify words to be set aside by a Match operation. Then Multi-Write a tag bit for all matching words. The set up for the next operation would include

writing a "1" in the AMMR and a "0" in the AMCR for the tag bit, so that no words previously tagged in this bit position could be set to a Match State.

The use of linking is through a successful MATCH operation, and the appropriate setting of link bits in AMCR. The setting of link bits L_1 and L_2, for word n in the memory may be accomplished through a match on word n, word n+1, or word n-1, depending upon the contents of the Link Bit Register. This operation is, thus, dependent on programmed control, and has been included in the Associative Unit by considerations which were found to be significant in the text processing and pattern classification studies.

MATCH SELECT is a signal which polls the word logic for each word in succession, until a word in the (1, 0) state is sampled. The first word in the (1, 0) state is set to the (0, 1) state and a response is transmitted to set the MATCH SET flip flop, transmitting a signal to the AU control unit. The time to find the first word which matches is, therefore, primarily the transmission delay in propagating the MATCH SELECT test pulse plus the time to turn on the flip flop. MATCH TEST is similar to the MATCH SELECT; it polls the word logic until a word in the (1, 0) state is found and, then provides a response indicating that a match does exist in at least one memory word. MATCH TEST does not alter the state of any word.

SUMMARY

The design of the associative processor has been carried to the point where a conceptual description has been presented, the basic operations (or "machine

language") have been defined, and the logic at the memory word level has been indicated. By drawing on the experience acquired in specifying machines to solve selected non numeric problems, decisions on functions to be included in the associative processor have been made with a view to enhancement of non numeric processing capability. This capability is required for certain intelligence system functions. Thus, the basis has been established by which a hardware implementation could proceed. The choice of technology for the associative memory elements is left open, since the state-of-the-art is dynamic, and promising developments may be imminent in both magnetic technology[1] and integrated circuits.[2,3] Nevertheless, the design has been specified sufficiently so that logical design would follow by the use of state-of-the-art design techniques, from the description of the logic at the memory word level.

The complete design of the associative processor would require many more details than presented in this report. The microinstruction format, as given in Figure 25 is not completely specified and lacks detail which would indicate all of the possible ways to transfer data in preparation for executing a sequence of basic associative operations. The complete specification of the machine and its use would also require detailed description on the generation of microroutines and

1. Rajchman, J. A., "Memories in Present and Future Generations of Computers", IEEE Spectrum, Vol. 2, No. 11 pp 90-96; 1965.

2. Phillips, A. B., "Monolithic Integrated Circuits," IEEE Spectrum, Vol. 1, No. 6 pp 83-102; 1964.

3. Richmond, W. H., "Integrated Circuits for Commercial Computers," Datamation, Vol. 11, No. 11, pp. 29-33; 1965.

Section IX

CONCLUSIONS AND RECOMMENDATIONS

Six associative processor designs have been presented; five of these are special-purpose processors designed to include features which optimize processing in certain selected problem areas, and one is a general-purpose associative processor. All designs included at least one associative memory and the capability for parallel processing by performing the same operation simultaneously on all words in the memory. The emphasis throughout the study was on the characteristics and significant features of the problems, and a unique processor was designed for each problem. However, it was possible to identify characteristics which the different designs possess in common, and these characteristics are included in the general-purpose processor.

The most important difference in concept, in the transition from individual special-problem processors to the general-purpose associative processor (covered in Section VIII), is the departure from the view that a general-purpose computer is associated as a controlling computer with an associative memory. Instead, a relatively autonomous Associative Unit is proposed, containing its own "stored-in" program sequences to guide its operations through a set of complex tasks. The controlling General Processor Unit requires minimal time for communication with the Associative Unit, generally at the start and completion of a task, but not during performance of a task by the Associative Unit.

In the design of an associative file processor unit (see Section V), it was recognized that the most significant gains could be made by devising means which minimize the amount of data transfer to a central processor. The capacity of the bulk store was considered to be of the order of 80 million 8-bit characters, i.e., much greater than would be reasonably assumed for an associative memory but representative of the capacity required for data files in military command and intelligence systems. In conventional systems, techniques have been developed for directly positioning the access mechanism to read records that contain data to be tested for values of specified parameters (or fields) and combinations of these parameters, which are usually expressed in terms of Boolean connectives. However, a large number or records must be transferred from the bulk file to the CPU (which makes the logical and numeric checks to select those qualifying as responses to a query). In some cases, the amount of data transferred in searching can be reduced by indexing and chaining techniques. However, such techniques reduce overall system efficiency by substantially increasing the time required to add new records. The associative file processor retains the capability for using indexes in the CPU by which the disk access mechanism is positioned so that only records relevant to the query will be read, but associative circuitry has been designed so that field values in the records are compared with query parameters while reading the records. Thus, it is possible to select only qualifying records for transmission to the CPU. This results in a very significant saving in transmission time, usage of CPU random-access memory, and CPU processing time.

The results of the formatted file processor study have importance beyond the objective of designing a general-purpose associative processor. A bulk file unit of the type described would be a significant benefit to a conventional data processing system as well as to an advanced design; it might be regarded as an "intelligent channel" when compared with present-day usage and access to bulk files via computer data channels.

The problem of pattern classification (discussed in Section VI) was studied with the object of deriving the mathematical formulation of the problem in terms of operations which, if implemented in hardware, would facilitate solution of the problem. Measures of "similarity" derived in various independent efforts were considered as models of the classification process, and their defining parameters were put on a common basis. This was possible despite the fact that the different models cannot be derived from one another. The required capabilities were found to be:

1. Ability to rapidly calculate sum and intersection functions of vectors with binary components,
2. Ability to rapidly find subsequences of vectors which match given sequences.
3. Ability to compute rapidly the inner product function of two vectors having positive integers as components.

While it is considered reasonable to implement the second capability in hardware, it is very doubtful that the cost of hardware implementation of the other two functions would be justifiable. In the final design, an algorithmic method of obtaining the first and third functions is recommended.

Significant increases in operating speeds are possible in the implementation of an associative processor for the solution to the data extraction problem. It was observed in Section II that the problem is logically divided into three functional segments: Text Input, Text Search, and Formatting and Output. The first of these is concerned with the initial editing of the text as it is received in the processor, the second covers the area of the actual extraction of the data from the text, and the third relates to the process of formulating the actual output in the manner desired by the user.

The gains to be made in the parallel processing of textual data result from the feasibility of searching in parallel for several items similar in format. Additional gains are possible when one considers processing more than one similar document at a time. The system throughput time for documents other than the first few in any batch would be diminished, since the performance of later functions could be overlapped with performance of input functions.

At a slightly more detailed level, the processor is organized in a manner to facilitate the processing of strings of characters using the shift capability of the mask and compare registers. The organization of the storage into single character cells further reflects the character-oriented nature of the problem. The examination of characters and strings of characters may proceed at the character level or at the bit level by means of the mask capability designed into the processor. This imposes no arbitrary restriction on the type of manipulation available to the user of this processor and opens the way to extensive manipulations of the data not easily done on more conventional organizations.

The design of an associative processor for dictionary search processing (described in Section III) was based upon the premise that major gains would be possible if the effective data processing rate in the processor could be increased. The design results from the consideration of the data flow in such a system. Significant gains result from the parallel searching by the memory unit involved and from the technique of performing, in parallel, the separate functions of different phases of the process. The properties of the text yield clues to the solution of the second part of the problem in that a large proportion of the text with respect to the actual terms used is redundant. This is because a small percentage of unique words (types) account for a large percentage of total words (tokens) in narrative English text. Samples studied indicate that less than 1 percent of the unique words (e.g., THE, OF, A, FOR) account for 45 to 50 percent of the total words; 99 percent of the unique words represent the remaining 50 to 55 percent of the text.

This technique of parallel processing by means of an extra, high-speed module in the processor can yield a gain of 50 percent in the overall processing time for the dictionary task. This decrease in the processing time will be in addition to the gains realized from the associative properties of the processor embodied in the main processing element. The suggestion for processor design is that dictionary search be done in two phases: a small high-speed memory for the 1 percent of the unique words and a larger memory for the remaining 99 percent.

The memory unit of the input processing module as well as that of the dictionary processing module has the capability of linking adjacent registers of

the memory during the search. This allows the programmer to ignore the problem of packing his data in some special code in order that the data will fit compactly in one register. The requirement for algorithmic retrieval or storage of an actual address of the corollary information desired is eliminated by this feature. Thus, additional information about some word can be stored in the next highest register in the memory, and the program merely "links higher" on exact match to be able to extract this information without having to know its address in the memory.

The design of the associative processor for the compilation of statistics on textual material (covered in Section IV) is related in organization and function to the dictionary processor. The relation of processing speed between the two subprocessors is similar in each of the above processors and is so for the same reasons. The associative processor is therefore organized to take advantage of the same potential time savings found in the dictionary processor. The design is oriented to take advantage of the possible reduction in data flow to the central processor and at the same time to accomplish this reduction in data flow in parallel with the operations of the central processor. Thus, the design allows for the word breakup processing and identification of the highest-frequency words in parallel with the looking up of words already found and placed in the dictionary. The preprocessing of the text in the first processor results in a decreased data flow in the second processor. Since the first processor is also independently programmed, the table of special words and definitions of word boundaries may be changed to reflect the characteristics of the current data sample.

The results of several studies thus indicate that the concept of one or more small general-purpose associative processors of varying speeds and capabilities located in the data flow path would be highly desirable. This is evident in the overall simplification of the programming of the system that results, and in the overlapping and consequent net decrease in the number of operations that each processor is required to execute for this task.

The results of the study of the associative processing designs and techniques applied to non-numeric data processing indicate that there exist several areas of investigation requiring further study. The development of the concept of the associative channel processor, with the resultant savings in operation times for file processing, is a prime area for additional design effort leading to the construction of a feasibility model. This model would use current technology and could be implemented for use with a variety of general-purpose processors. The adaptation of this concept for use with current hardware does not preclude its use with associative hardware of the future. Such processors, described in Section VIII, will enable the search procedure to be made even more effective than it is in conjunction with the current hardware available as a general-purpose computer. This improvement can be accomplished by means of more sophisticated techniques for analyzing the queries to be sent to the channel processor for search of the file. Aspects of the queries that are parallel can be detected currently with very laborious techniques, which would cut down on the overall speed increase available from the use of this device with a conventional processor.

It is evident that, while the associative file processor is being developed, considerable gains are possible from the study of advanced problems and the techniques for their solution both on current equipment and on associative and non-associative equipment of the future. Using extensions of the problem models (as stated in Sections II through V), techniques can be developed for exploiting information inherent in the data that it is not now practical to extract. The problem of automatic data extraction is currently critical in a number of areas and will become more critical when optical character readers are perfected and more widely used. An increasing number of formatted and semi-formatted reports are being produced and transmitted, with an increasing need for techniques to extract data from these reports for storage and display.

Because processing requirements on current machines are excessive, they appear to be a prime candidate for the application of, and techniques developed for, associative hardware. Together with the associative channel processor mentioned above, this hardware would add capability to large formatted file systems and improve their effectiveness.

APPENDIX I

ADDITIONAL INSTRUCTIONS

LINK RIGHT

Each bit position of the compare register corresponding to a "1" bit in the mask register is compared to storage. A link bit is set to one in the next adjacent registers to those registers where all "compared" bits matched. If the reset bit in the instruction is 1 and a mismatch occurs, the link bit is set to zero, otherwise the link bit is unchanged on a mismatch. A corresponding bit may be set or cleared in the matched register by inclusion of another one bit in the instruction.

Format: LRi, j

LINK LEFT

Each bit position of the compare register corresponding to a "1" bit in the mask register is compared to storage. A link bit is set to "1" in the preceding adjacent registers to those registers where all "compared" bits matched. If the reset bit in the instruction is 1 and a mismatch occurs, the link bit is set to zero, otherwise the link bit is unchanged on a mismatch. A corresponding bit may be set or cleared in the matched register by inclusion of another one bit in the instruction.

Format: LLi, j

INTERSECT — Each bit position of the compare register corresponding to "1" bits in the mask register is compared to storage logically. A pulse is generated in the first counter position for each matching position in the memory register. A delay is made for each storage bit. No pulses are generated when corresponding positions do not match.
Format: IN

NEAREST MATCH — The contents of the compare register corresponding to "1" bits in the mask register are compared numerically to each storage register. A match bit in the register(s) having the smallest numeric difference is (are) set to one. Bit positions of storage corresponding to "Ø" bits in the mask register are ignored.
Format: NM

MATCH GREATER — The contents of the compare register corresponding to "1" bits in the mask register are compared numerically to each storage register. A match bit is set to one in each storage register exceeding the compare register. Bit position of storage corresponding to "Ø" bits in the mask register are ignored.
Format: MG

MATCH LESS — The contents of the compare register corresponding to "1" bits in the mask register are compared numerically to each storage register. A match bit is set to one in each storage register exceeded by the compare register. Bit positions of storage corresponding to "Ø" bits in the mask register are ignored.
Format: ML

CLEAR COUNTERS — The counters in memory whose corresponding match bit(s) is (are) set are set to zero. When no bit(s) is specified in the instruction, all counters are set to zero. (main memory only)
Format: $CCN_1, N_2, \ldots, N_k$

INNER PRODUCT — The bits of each character position of the comparison register corresponding to "1" bits in the mask register are multiplied against the corresponding character position in small storage. The resultants are accumulated in accumulator registers associated with each storage position. (Small memory only)
Format: IP

TRANSMIT — N consecutive cells of storage are transmitted from main storage to high speed storage. Transmission begins at the first cell in which the T-match bit is set

and continues for the smaller of the following numbers of cells; the number N, specified in the instruction, or the number K, of cells available in the high speed memory. Starting position in high speed memory is specified in the instruction. The T-match bit is turned off in main memory when data transfer commences.

Format: T, N, K

CLEAR ACCUMULATORS

The accumulators in memory whose corresponding bits are set are set to zero. When no bits are specified in the instruction, all accumulators are set to zero. (Small memory only)

Format: CA, N_1, . . . , N_k

BIBLIOGRAPHY

Ahrons, R.W., Burns, L. L. Jr., "Superconductive Memories", Computer design, p. 12, Jan 64.

Air Force Systems Command, "Implicitly Programmed Systems Working Group," Vol. II, USAF Exploratory Development Program In Information Processing Technology, AFSC, Wash. D.C., AD 458 660, DIV 32/1, Oct 64.

Alexander, D. C., Dennard, R. H., Post, F. L., "A Delay Line Approach To Associative Memory," IBM Advanced Systems—Endicott, N.Y., 17.022, May 61.

Almendinger, V. V., "Span Reference Manual (System Operation) (Statistical Processing and Analysis)," SDC, Santa Monica, Calif., AD 613 284, 30/1, 32/1, Feb 65, 33 pp.

Almendinger, V. V., "Span Reference Manual Data Files Manipulation and Processing," SDC, Santa Monica, Calif., AD 613 289, DIV 30/1, 32/1, Mar 65, 92 pp.

Amdahl, G. M., "New Concepts In Computing System Design," IBM Research, RC-526, IBM CONFIDENTIAL, 5 Aug 61.

Amdahl, G. M., Blaauw, G. A., Brooks, F. P. Jr., "Architecture of the IBM System/360," IBM Journal of Research and Development, Vol. 8, No. 2, Apr 64.

Anderson, J. L., "Search on Range Associative Memory" IBM Tech. Disclosure Bulletin, Vol. 5, No. 5, Disclosure #75,627, Oct 62.

Anderson, J. P., "A Computer For Direct Execution of Algorithmic Languages," Proc. EJCC, 61.

Anderson, J. P., et al, "D825 - A multiple-Computer System for Command and Control," Proc. FJCC, 62.

Aoki, M., Estrin, G., "The Fixed-Plus-Variable Computer System In Dynamic Programming Formulation of Control System Optimization Problems - Part I," RPT. No. 60-66, UCLA, May 61, 33 pp.

Auizienis, A., "Signed-Digit Number Representations for Fast Parallel Arithmetic," IRE. Transaction on Electronic Computers, Vol. EC-10, pp. 389-400, Sept 61.

Baldwin, F. R., et al, "A Multiprocessing Approach To A Large Computer System," IBM Systems Journal, Sept 62.

Ball, J. C., et al, "On The Use Of The Solomon Parallel Processing Computer," FJCC, 62.

Barbieri, R., "Computer List Processing Languages," IBM Data Systems, Poughkeepsie, New York, TR 00.1209, 11 Nov 64.

Barnard, J. D., Behnke, F. A., Linquist, A. B., Seeber, R. R., "Structure of a Cryogenic Associative Compiuer," IBM Data Systems, Poughkeepsie, New York, TR 00.1050, IBM CONFIDENTIAL, 23 Sep 63.

Barnum, A. A., Knapp, M. A., "Computer Organization," spartan Books, 63.

Beatty, J. C., "On Some Properties of the Semi-Group of a Machine Which Are Preserved Under State Minimization," IBM Research, RC-1199, 22 May 64.

Beatty, J. C., Muroga, S., "File Memory Addressing," IBM Research, RC-1282, IBM CONFIDENTIAL, 18 Sep 64.

Behnke, F. A., Plonsky, A. T., "Associative Storage Techniques," IBM Kingston, TP 61-1376, AF-30(602)2161.

Behnke, F. A., Plonsky, A. T., "Associative Storage Technology," IBM Kingston.

Behnke, F. A., Rosenberger, G. B., "Cryogenic Associative Processor," Final Report, IBM-Kingston, New York, RADC-TDR-63-432, AF 30(602)2608, IBM CONFIDENTIAL, 3 Sep 63.

Belady, L. A., "Measurements on Programs - One Level Store Simulations," IBM Research, RC-1420, IBM CONFIDENTIAL, 15 Jun 65, 67 pp.

Bennett, R. W., Julius Berger, H. Y., "A Special Purpose Microprogram For The Preprocessing of Input Data," IBM Advanced Systems, Endicott, New York, 17-153, 65A00925-MF011, IBM CONFIDENTIAL, Nov 64.

Bledsoe, W., Browing, I., "Pattern Recognition and Reading by Machine," Proc. EJCC, 59, p. 225-232.

Bloom, L., Cohen, M., Porter, S., "Considerations in the Design of a Computer with High Logic-To-Memory Speed Ratio," Proc. Gigacycle Computing Systems Sessions, AIEE Winter General Meeting, Jan 62, p. 53.

Bobrow, D. G., Raphael, B., "A Comparison of List Processing Languages," COMM. ACM, Vol. 7, No. 7, Apr 64.

Branning, H. F., "Randam: Random Access Non-Destructive Advanced Memory," IBM Federal Systems, Owego, N.Y., 61-503-2, 61-RRA.

Brenda, J. G., "A Systematic Analysis of Equipment Selection and Data Allocation Strategies for Shared Computer Systems," IBM Advanced Systems, Endicott, New York, 17-158, Dec 64.

Brenza, J. G., Jackson, R. C.. Rhodes, W. H. Jr., Winger, W. D., "A Built-In Table Lookup Arithmetic Unit."

Brown, J. R. Jr., "A Semi-Permanent Associative Memory and Code Converter," Special Technical Conference on Nonlinear Magnetics, Los Angeles, Nov 61.

Bucholz, W., "File Organization and Addressing," IBM Systems Journal, Vol. 2, Jun 63.

Burroughs Corp., "The Descriptor, a Definition of the B-5000 Information Processing System," Burroughs Corp., Detroit, Michigan, Bulletin 5000-2000L-P, Feb 61.

Bussell, B., "Properties of a Variable Structure Computer System in the Solution of Parabolic Partial Differential Equations," PhD Dissertation, UCLA, Aug 62.

Bussell, B., Estrin, G., "Design of a Fixed Plus Variable Structure Computer for the Solution of a Diffusion Equation," Part I, UCLA, AD 263 883, DIU 15/1, 25/1, 30/1, Jul 61.

Carr, J. J., "Evaluation of Electronic Memories Including SHMOO Transfluxor Memory Device," Frankford Arsenal, AD 445 122 L DIU 8/1, 30/1; RESTRICTED DIST: Attn Frankford Arsenal, Phila.

Caschera, J., "Research on Ferret Associative Memory," Philco Corp., Willow Grove, Pa., AD 453 096, DIV 30/1, 31 Aug 64 - 30 Nov 64.

Chadurjian, F., "Comparator," U.S. Patent No. 2,973,508, Feb 61.

Chesarek, J., "The Stored Program Calculator, SPC—A Small High Performance Computer," IBM Advanced Systems, IBM CONFIDENTIAL, Oct 64.

Chu, Yaohan, "A Destructive-Readout Associative Memory," IEEE Trans. on Electronic Computers, Vol. EC-14, No. 4, p. 600, Aug 65.

Clark, W. A., Farley, B. G., "Generalization of Pattern Recognition in a Self-Organizing System," Proc. WJCC, 55, pp. 86-91.

Comfort, W. T., "A Modified Holland Machine," Proc. FJCC, 63.

Comfort, W. T., "Highly Parallel Machines," IBM Owego, IBM Tech Rpt. 62-825-496, Oct 62.

Comfort, W. T., "Multiword List Items," IBM Owego, IBM Tech. Rpt. 61-907-198, Nov 61.

Computer Command and Control Co., "Applic ition of Associative Memoi ies to the Weapon Assignment Problem of NTDS," Computer Command and Control Company, ONR Report No. 13-101-8 (SECRET), Contract # #NOnr 4068(00), Naval Analysis Group, SECRET.

Computer Command and Control Co., "Associative Memory Computer System Description and Selected Naval Applications," Computer Command and Control Co., 10 Apr 65.

Computer Comm nd and Control Co., "Summary of Investigation n Associative Memories " Rpt. No. 5, Computer Command and Control Co., 15 Jan 64.

Computer Command and Control Co., "Pattern Recognition Process for Bubble Chamber Pictures," Computer Command and Control Company, ONR Report No. 2-102-2, Contract # NOnr 4068(00), Naval Analysis Group.

Conway, M. E., "A Multiprocessor System Design," Proc. FJCC, p. 139, Nov 63.

Corbell, R. C., "Tunnell Diode Associative Memory," Master's Thesis UCLA, Jun 62.

Corneretto, A., "Associative Memories, A Many-Pronged Design Effort," Electronic Design, p. 40, 1 Feb 63.

Corneretto, A., "3-K BIT Associative Memory Works at Room Temperature," Electronic Design, 5 Jul 62.

Cr. chlow, A. J., "Generalized Multiprocessing and Multiprogramming Systems," Proc. FJCC, 63.

Danylchuck, I., Pernesni, A. J., Sagal, M. W., "Plated Wi Magnetic Film Memories," Intermag Proceedings, Washington, D. C., Apr 64.

Davies, P. M., "A Superconductive Associative Memory," Proc. SJCC, May 62, p. 79.

Davies, P. M., "Design of an Associative Computer," Proc. Pacific Computer Conference, Pasadena, California, Mar 63, p. 109.

Davies, P., "The Associative Computer," Proc. Pacific Computer Conference, California Institute of Technology, 15 Mar 63.

Deronald, C. H., Fotheringham, J. A., "The ALTAS Computer," Datamation, May 61.

Dreyfuss, P., "System Design of the Gamma 60," Proc. WJCC, 58.

Duda, W. L., Elfant, R. F., "Electronically Addressable Bulk Memories," IBM Research, Yorktown Heights, New York, RC-1250, IBM CONFIDENTIAL, 10 Aug 64.

Dunham, B., North, J. H., "The Problem of Selecting Logically Efficient Building Blocks and Hookups," IBM Research, Yorktown Heights, New York, RC-785, 18 Sep 62.

Elfant, R. F., "Design and Cost Estimate for Electronically Addressable Bulk Memory," IBM Research, Yorktown Heights, New York, RC-1292, IBM CONFIDENTIAL, sep 64.

Elmendorf, W. R., "Program Sequencing and Storage Addressing in a Multi-Processing System," IBM Research, Yorktown Heights, New York, RC-501, IBM CONFIDENTIAL, 4 Aug 61.

Erikson, A. J., "Magnetic Thin-Sheet Memory," Quarterly Report, RCA Defense Electronics Products, Camden, N.J., AD 460 381, DIV 30/1, 17/1; 1 Jun 64-30 Sep 64.

Erickson, D. K., Hughes, J., Turner, R. L., "Information Storage and Retrieval System," IBM General Products, San Jose, Calif., PT-2909-F, IBM CONFIDENTIAL, Apr 65.

Estrin, G., Fuller, R., "Algorithms for Content Addressable Memory Organizations," Proc. Pacific Computer Conference, Pasadena, Calif., Mar 63.

Estrin, G., "Organization of Computing Systems — The Fixed Plus Variable Structure Computer," Proc. WJCC, San Francisco, Calif., 3-5 May 60, pp 33-40.

Estrin, G., Fuller, R., "Some Applications For Content-Addressable Memories," Proc. FJCC, Las Vegas, Nev., Nov 63.

Evans, J., Florkowsky, J. H., "A Correction Scheme to Use Imperfect Memory Array With No Reduction In Speed," IBM Advanced Systems, IBM CONFIDENTIAL, 20 Mar 64.

Evans, J., Florkowsky, J. H., "Multiple Addressing for Fixed-Tag Associative Memories," IBM Advanced Systems, Endicott, New York, TR-17-138, IBM CONFIDENTIAL, 17 Jan 64.

Ewing, R. G., Davies, P. M., "An Associative Processor," Proc. FJCC, Vol 26, Part I, Oct 64, p. 147.

Falkoff, A. D., "Algorithms For Parallel Search Memories," IBM Research, Yorktown Heights, New York, RC-533, Aug 61.

Falkoff, A. D., "Algorithms for Parallel Search Memories," IBM Research, Yorktown Heights, New York, RC-658, IBM CONFIDENTIAL, Apr 62.

Falkoff, A. D., "Algorithms for Parallel Search Memories," Journal of the ACM, Oct 62, P. 488.

Falkoff, A.D., "Formal Description of Processes—The First Step In Design Automation," IBM Research, Yorktown Heights, New York, NC-510, 15 Jun 65.

Falkoff, A. D., "Program Sequence Control In A Multiprocessing System Using Associative Storage," IBM Advanced Systems, Mohansic, New York, Technical Memo No. 15, IBM CONFIDENTIAL, 28 Sep 60.

Fan, G., Mee, C. D., "A Beam Addressable Memory File," RC-1346, IBM Research, Yorktown Heights, New York, RC-1346, IBM CONFIDENTIAL, 7 Jan 65.

Farber, D. J., et al, "Snobol, A Strink Manipulation Language," J. ACM, Vol. II, No. 2, Jan 64.

Farrar, J. M. Jr., Courtney, R. H. Jr., "Associative Memory Applications for Intelligence Data Processing," IBM Federal Systems Division, Rockville, Maryland, IBM CONFIDENTIAL, 29 Dec 61.

Feldman, J. A., "Aspects of Associative Processing," Lincoln Lab, MIT, Lexington, Mass., AD 614 634, DIV 30/1, Apr 65.

Ferris, Ronald J., "An Analysis of the Multiple Instantaneous Response File," RADC, Griffiss AFB, N.Y., AD 610 131, DIV 32/1, 30/1; Dec 64.

Fildes, J. J., Zeitler, G. Jr., "A Memory Address Controller for Tele-Processing Systems," IBM Data Processing, Poughkeepsie, New York, 30 Apr 62.

Fleisher, H., "Combinatorial Techniques for Performing Arithmetic and Logical Operations," IBM Research, Yorktown Heights, New York, RC-289.

Flynn, M. J., "Operations In an Associative Memory," PhD Thesis, Purdue University, BTP-62-1782, APFILS, Jun 61.

Flynn, M. J., Machol, R. E., "Logical and Functional Specification of an Associative Memory," IBM Data Systems, TR 00.852, 15 Feb 62.

Frate, G. J., Stromick, S., "Compendium of Storage and Retrieval Devices and Techniques," RADC, AD 450 182L, DIV 30/1, RESTRICTED DIST: RADC ATTN EMIIG, Sep 64.

Frei, E. H., Goldberg, J., "A Method for Resolving Multiple Responses In a Parallel Search File," IRI Transactions on Electronic Computers, Vol. EC-10, No. 4, Dec 61, pp. 718-722.

French, W. K., "Associative Memory", IBM Patent, U. S. Patent No. 3,123,706, 3 Mar 64.

French, W. K., "Associative Memory," U. S. Patent No. 3,131,291, Apr 64.

Fuller, H. W., McCormack, T. L., "Study and Investigation of Technique for Constructing Medium-Speed Random Access Mass Memory, LFE Electronics, Boston, Mass., AD 614 821, DIV 30/1, 14/1, 25/1, 26/1; Mar 65.

Fuller, R. H., "Content-Addressable Memory Systems," UCLA, AD 417 644, DIV 30/1, Jun 63.

Fuller, R. H., Bird, R. M., Medick, J. N., "Associative Processor Study," Librascope Div., Glendale, Calif., AD 608 427 DIV 30/1, Oct 64.

Fuller, R. H., Estrin, G., "Some Applications for Content-Addressable Memories," Proc. FJCC, Nov 63, p. 495.

Fuller, R. H., Salzer, J. M., "Associative Processor Study," General Precision Inc., AD 608 427, Oct 64.

Futami, K., et. al., "The Plated-Woven Wire Memory Matrix," Interman Proceedings, Washington, D. C., Apr 64.

Gall, R. G., "A Hardware-Integrated GPC/Search Memory," Proc. IFIPS 1964, FJCC, 64.

Gelernter, H., "System Requirements of a Digital Computer for the Manipulation of List Structures," IBM.

Goldberg, J., Green, M. W., "Large Files for Information Retrieval Based on Simultaneous Interrogation of All Items," Proc. Symposium on Large-Capacity Memory Techniques, May 61, p. 63.

Goldstine, Horwitz, Karp, Miller, "On the Parallel Execution of Macro Instructions," IBM Research, Yorktown Heights, New York, RC-1262, IBM CONFIDENTIAL, 17 Aug 64.

Goodyear Aircraft Corp., "Collection of Notes on Associative Memory," Report No., GER 10857, Akron, Ohio, Oct 62.

Green, B. F., "Computer Languages for Symbol Manipulation," IRE Transactions on Electronic Computers, Dec 61.

Greene, J. E., Dean, R. F., Updike, B. M., "Microprogrammed Implementation of the IBM System/360-30 Machine Organization," General Products, Endicott, New York, TR-01.813, Jun 64.

Gregory, J., McReynolds, R, "The Solomon Computer," IEEE Trans. On Electronic Computers, Vol. EC-12, No. 5, Dec 63.

Griffith, J. E., "An Intrinsically Addressed Processing System," IBM Systems Journal, Vol. 2, Sept-Dec 63.

Griffith, J. E., "Table Lookup Computers," IBM Data Systems, Kingston, New York, TR-21.053, IBM CONFIDENTIAL, 14 Mar 62.

Griffith, J., "Techniques for Advanced Information Processing Systems," First Congress on the Information System Sciences, Hot Springs, Va., 18-21 Nov 62.

Hawkins, J. K., "Self-Organizing Systems—A Review and Commentary," Proc. IRE, 1961, Vol. 49, pp 31-48.

Hellerman, H., "A Compiler and Machine Organization for Parallel Processing of Algebraic Expressions," IBM Advanced Systems, Endicott, New York, Report No. 17-150, IBM CONFIDENTIAL, Dec. 64.

Hellerman, H. H., "A Directory Control System for Multiprogramming," IBM Research, Yorktown, New York, RC-1095, Oct 63.

Hellerman, H., "Experimental Personalized Array Translation System," Comm. ACM, Vol. 7, No. 7., 64.

Hellerman, H., "On the Organization of a Multi-Programming—Multiprocessing System," IBM Research, Yorktown Heights, New York, RC-522, IBM CONFIDENTIAL, 5 Sep 61.

Herwitz, P. S., "Harvest System," IBM Research, Yorktown Heights, New York, RC-64, Nov 58.

Holland, J. R., "A Universal Computer Capable of Executing an Arbitrary Number of Sub—Programs Simultaneously," Proc. EJCC, Boston, 1-3 Dec 59.

Holland, J., "Interative Circuit Computers," Proc. WJCC, 60.

Holum, B. A., "Some Uses of an Associative Memory As a Real-Time Control," IBM Corporation, New York, New York, SRI Term Paper No. 11-31, IBM CONFIDENTIAL, Apr 64.

Hughes Aircraft Co., "A Proposal for the Study of Associative Processing Techniques," Rpt No. FP 63-16-276, Hughes Aircraft Co., 14 Oct 63.

Hunt, R. T., Snider, D. L., Suprise, J., Boyd, H. N., "Study of Elastic Switching For Associative Memory Systems," Good Year Aircraft Corp, DDC, Feb 64.

IBM, "Associative Processing Techniques," IBM Proposal to RADC RTD, GRIFISS A. B, RFP # 64-423, 9 Oct 63.

IBM Advanced Systems Division, "A Proposal for the Study of Advanced Information Retrieval Techniques," IBM Advanced Systems, Yorktown Heights, New York, IBM Proposal to USA/SSA, Ft. Monmouth, 30 Mar 62.

IBM Data Processing Division, "File Organization Techniques for Direct Access Storage Devices," IBM Data Processing, Mechanicsburg, Pa., IBM CONFIDENTIAL.

IBM Federal Systems Division, "An Associative Memory Using Superconductive Techniques," IBM Federal Systems, Rockville, Maryland, TP60-3500 to RADC, 13 Sep 60.

IBM Federal Systems Division, "Associative Processor," IBM Proposal to RADC-AFSC, GRIFFISS AFB, 19 Jun 64.

IBM Federal Systems Division, "Cryogenic Associative Memory Techniques," IBM Proposal to RADC GRIFFISS, FSD-Rockville, AFPI-465 (d) (1).

IBM Federal Systems Division, "Hybrid Associative Computer Study," IBM Proposal to RADC, Rome, New York, 29 Jun 64.

IBM Federal Systems Division, "Study of the Applications of Parallel Search Memories," IBM Proposal to AFSC-ESD HANSCOM Field, FSD Rockville, ES-3-438L-3267/SBM, 29 Mar 63.

IBM Federal Systems Division, "Logical Memory Study," IBM Rockville-AF Cambridge Research Center, 7 Nov 60.

Iverson, K. E., "Formalism In Programming Languages," COMM. ACM, Vol. 7, No. 2, Feb 64.

Iverson, K.E., "Recent Applications of a Universal Programming Language," IBM Research, Yorktown, New York, NC-511m 15 Jun 65.

Johnson, L. R., "An Indirect Chaining Method for Addressing on Secondary Keys," Communications of the ACM, May 61.

Johnson, L. R., "On Operand Structure, Representation, Storage and Search," IBM Research, Yorktown, New York, RC-603, 5 Dec 61.

Johnson, L. R., McAndrew, M. H., "On Ordered Retrieval From an Associative Memory," IBM Journal of Research and Development, Vol. 8, No. 2, Apr 64, p. 189.

Joseph, E. C., Kaplan, A., "Target-Track Correlation with a Search Memory," Proc. 6th National Conv. on Military Electronics, Jun 62, p. 255.

Jutzi, W., "The Magnetic Pulse Field of the Bit Line in the Megabit Thin Magnetic Film Memory," IBM Research, Yorktown, New York, RZ-150, IBM CONFIDENTIAL, 22 Sep 64.

Kantariya, G. V., "Parallel Micro-Programming and the Principles of Design of Central Control Units for Digital Computers," AERO Space Technology Div., Library of Congress, AD 299 737, DIV 30/1, Dec 62.

Kaplan, A., "A Search Memory Subsystem for a General Purpose Computer," Proc. FJCC 1963, p. 193.

Kilburn, T., et al, "One Level Storage System," IRE Transactions on Electronic Computers, (also PGEC Apr 62), Apr 62.

King, G. W., "Table Lookup Procedures in Data Processing," IBM Research, Yorktown, New York, NC-166.

Kiseda, J. R., "A 128-Word, 36-Bit Magnetic Associative Memory," IBM Research, Yorktown, New York, NC-358, IBM CONFIDENTIAL, 17 Mar 64.

Kiseda, J. R., Peterson, H. E., Seelbach, W. C., Teig, M., "A Magentic Associative Memory," IBM Journal of Research and Development, Vol. 5, No. 2, Apr 61.

Koener, R., "A Memory Array Searching System," U.S. Patent No. 3,031,650, Apr 62.

Kolsky, H. G., "General Description of a Low-Cost Computer Organization Based on the Dynamic Streaming of Records," IBM Research, Yorktown, New York, RJ-210, 23 Feb 62.

Kochen, M., "Experimental Study of 'Hypothes-Formation' by Computer," IBM Research, Yorktown, New York, RC-294, 25 May 60.

Kovalick, V. E., "Literature References on: Modular Computer Organization, Parallel & Multi-Processor Computers," IBM Federal Systems, Owego, New York, IBM CONFIDENTIAL, 14 Nov 62.

Kurtz, G., Neilson, R., Schiff, A., Smith, G., "Table Lookup Study Model," IBM Federal Systems, Kingston, New York, IBM CONFIDENTIAL, 3 Aug 62.

Ledley, R. S., "Organization of Large Memory Systems," Proc. Large-Capacity Memory Techniques for Computing Systems, May 61, p. 15.

Lee, C. Y., "Intercommunicating Cells, Basis for a Distributed Logic Computer," Proc. FJCC, Philidelphia, Pa., Dec 62.

Lee, C. Y., Paull, M. Y., "A Content Addressable Distributed Logic Memory With Applications to Information Retrieval," Proc. IEEE, Jun 63.

Lee, E. S., "Associative Techniques with Complementing Flip-Flops," Proc. SJCC, May 63, p. 381.

Lee, E. S., "Solid State Associative Cells," Proc. Pacific Computer Conference, Calif. Inst. of Tech., 15-16 Mar 63.

Leiner, A. L., et. al., "Concurrently Operating Systems," Proc. IFIPS, 59.

Lewin, M. H., "Retrieval of Ordered Lists From a Content-Addressed Memory," RCA Review, Vol. XXIII, No. 2, Jun 62, pp. 215-229.

Lewin, M. H., Beelitz, H. R., Rajchman, J. A., "Fixed, Associative Memory Using Evaporated Organic Diode Arrays," Proc FJCC, Nov 63, p. 101.

LFE Inc., "Study and Investigation of Technique for Constructing Medium-Speed Random Access Mass Memory," Laboratory for Electronics Inc., AD 614 831, Mar 65.

Lindquist, A. B., "An Application for a Small, Fast Associative Memory to Reduce the Access Time for Instructions in Loops," IBM Corporate, New York, New York, IBM Term Paper No. 4-39, 19 Dec 61.

Lindquist, A. B., "An Associative Local Store," IBM ITL Meeting on Machine Organization, IBM CONFIDENTIAL, Nov 63.

Lindquist, A. B., "Associative Memory for Highly Parallel System," Proc. FJCC, Las Vegas, Nev., 12-14 Nov 63.

Lindquist, A. B., "Coding of Trees for Use In an Associative Memory," IBM Corp., 3 Aug 62.

Lindquist, A. B., "Cryotron Associative Memory Cell," IBM Data Systems, Poughkeepsie, New York, 6SC-01774-MF004, Mar 65.

Lonergan, W., King, P., "Design of the B-5000 System," Datamation, May 61.

Long, T. R., "Electrodeposited Memory Elements for a NonDestructive Memory," Journal of Applied Physics, May 60, Supplement to Vol 31, No. 5, pp 123s-124s.

Louis, H. P., Shevel, W. L. Jr., "Storage Systems - Present Standards and Anticipated Developments," IBM Research, Yorktown, New York, RC-1222, 23 June 64.

Love, H. H., Savitt, D. A., "Associative Processing Techniques Study," RADC-TR-65-32, Hughes Aircraft, FR-65-11-18, AF30(602)-3279, AD-616-620 DIV 30/2, Apr. 1965.

Luckfield, W. J., "A Multiple File Organization for Information Retrieval Systems," TIE 6408-0857, IBM CONFIDENTIAL.

Lussier, R. R., Schneider, R. P., "All-Magnetic Content Addressed Memory," Electronic Industries, Mar. 63, p 92.

McAteer, J. E., Capobianco, J. A., Koppel, R. L., "Associative Memory System Implementation and Characteristics," Proc IFIPS, 1964 FJCC.

McCarthy, J., et al, "LISP 1.5 Programmers Manual, MIT Press, 1962.

McCormick, B. H., Divilbiss, J. L., "Tentative Logical Realization of a Pattern Recognition Computer," Digital Computer Laboratory, University of Illinois, Report No. 403.

McDermid, W. L., Peterson, H. E., "A Magnetic Associative Memory System," IBM Journal of Research and Development, Vol. 5, No. 1, Jan. 1961.

McInnes, D. S., "A Micro-Programming Approach to Optical Scanning," Item No. 6408-0844, KWIC Index to TIE.

Macklin, D., "Dual M2M - Multiprocessor Communication System," SRI Paper 9-37, Ch-NY, Aug 63.

Maher, R. J., "Probiems of Storage Allocation in a Multiprocessor Multiprogrammed System," Comm. ACM, Oct, 1961.

Mann, H. T., Rogers, J. L., "A Cryogenic 'Between-Limits' Associative Memory," Proc. Nat. Aerospace Electronics Convention, May 62, p. 359.

Meggitt, J. E., "A Character Computer for High Level Language Interpretation," IBM Systems Journal, Vol 3, No. 1, 1964.

Minnick, R. C., "Magnetic Comparators and Code Convertors," Symposium on the Application of Switching Theory in Space Technology, Sunnyvale, California, Feb. 62.

Mueller, O., "A Very Small, Very Low Cost Computer for Use as a Black Box Computer," IBM Research, Yorktown, New York, RC-1206, 3 Jun 64.

Mullery, A. P., "A Procedure-Oriented Machine Language," IEEE Transactions on Electronic Computers, Aug. 1964.

Mullery, A. P., et al, "ADAM - A Problem Oriented Symbol Processor," SJCC 1963.

Mullery, A. P., Rice, R., Schauer, R. F., "Specifications for ADAM and ABEL," IBM Research, Yorktown, New York, RC-631, IBM CONFIDENTIAL, 1 Mar 62.

Muroga, S., "Application of Group Theory and Coding Theory to File Memory Addressing," IBM Research, Yorktown, New York, RC-1025.

Muth, V. O., Scidmore, A. K., "A Memory Organization for an Elementary List Processing Computer," IEEE Transactions on Electronic Computers, June 1963.

Nelson, R. A., "Problems in Automatic Storage Allocation," IBM Research, Yorktown, New York, RC-601, IBM CONFIDENTIAL, 27 Nov 61.

Nelson, R. A., "An Experimental Data Processing Machine," IBM ITL Meeting on Programming, IBM CONFIDENTIAL, Dec 62, p 267.

Newell, A., "Information Processing Language V Manual," Prentice Hall, 1964.

Newell, A., Tonge, F. M., "An Introduction to Information Processing Language V," Comm ACM, Apr. 1960.

Newell, A., "On Programming A Highly Parallel Machine to be an Intelligent Technician," Rand Corp., Santa Monica, Calif., AD 616 585, DIV 30/2, 1 April 1960.

Newell, A., "A Note On the Use of Scrambled Addressing for Associative Memories," Unpublished Paper, Dec. 1962.

Newhouse, V. L., Fruin, R. E., "A Cryogenic Data Addressed Memory," Proc. SJCC, May 1-3, 1962, p. 89.

Newhouse, V. L., Fruin, R. E., "Data Addressed Memory Using Thin-Film Cryotrons," Electronics, 4 May 62, p. 31.

Newman, E. G., Winter, L. F., "Magnetically Controlled Variable Logic," IBM Data Systems, Kingston, New York, IBM Journal of Research and Development Vol. 8, No. 3, July 1964.

Nolan, J. F., Armenti, A. W., "An Experimental On Line Data Storage and Retrieval System," Lincoln Labs Inst. of Tech., 3 Feb 1965, Lexington, Mass., AD 615 658, DIV 30/2, 32/3.

O'Neil, R. W., "Simulation of Some Multi-Processing Systems," IBM Research, Yorktown, New York, RC-824, IBM CONFIDENTIAL, 23 Oct 62.

O'Neil, R. W., "Some Notes on the Absolute Core Location Problem," IBM Research, Yorktown, New York, RC-600, IBM CONFIDENTIAL, 3 Jan 62.

Perkins, R., McGee, W. C., "Programmed Control of Multi-Computer Systems," Proc. IFIPS, 1962.

Perlis, Thornton, "Symbol Manipulation by Threaded Lists," Comm ACM, Vol. 3, No. 4, April 1960.

Petersen, H. E., "Content Addressing and Information Retrieval," IFIPS Congress '62, Munich, Germany, Aug. 1962.

Petrick, S. R., "Use of a List-Processing Language in Programming Simplification Procedures," AF CRL, Bedford, Mass., AD 273 759, DIV 30/1.

Philco, "Research on Ferret Associative Memory," Philco Corp., Willow Grove, Pa., AD 458 798, DIV 30/1, 30 Nov 64 - 28 Feb 65.

Phister, M., "Logical Design of Digital Computers," John Wiley & Sons. New York, New York, 1958, pp 144-167.

Plonsky, A. T., "Tag-Ordered Associative Memory," IBM Tech. Disclosure Bulletin, Vol 5, No. 8, Disclosure No. 131,099, Jan. 1963.

Porter, R. E., "The RW-400—A New Polymorphic Data System," Datamation, Jan/Feb 1960.

Pritchard, J. P., Jr., "Fabrication and Testing of Cryogenic Associative Memory Planes," Texas Instruments, Inc., Dallas, Texas, AD 616 491, DIV 30/2 25/6, 5 May - 13 Dec 64.

Pritchard, J. P., Jr., Wald, L. D., "Design of a Fully Associative Cryogenic Data Processor," Proc. of the International Conference on Nonlinear Magnetics, Apr. 64, p 2-5-1.

Prywes, N. H., ct al, "Interactions of Computer Language and Machine Design," Final Report, RADC-TDR-62-584, AD 292 033, Oct. 1962.

Prywes, N. S., Gray, H. J., "The Organization of a Multilist-Type Associative Memory," IEEE Transactions on Communications and Electronics, No. 68, Sep 63.

Raffel, J., Crowther, T. S., "A Proposal for an Associative Memory Using Magnetic Films, AD 612 832, Div 32, 25.

Raffel, J. I., Crowther, T. S., "A Proposal for an Associative Memory Using Magnetic Films," Lincoln Lab, MIT, Lexington, Mass., AD 454 538 Div 30/1, Mar. 64.

Rajchman, J. A., "Computer Memories: A Survey of the State of the Art," Proc. IRE, Vol. 49, Jan. 1961, pp 104-127.

RCA, "High Speed Data Processor System Research Project Lightning," RCA Electronic Data Processing Division, Camden, New Jersey, AD 240 499, Dec. 1959.

RCA, "Magnetic Thin-Sheet Memory," RCA - Defense Electronics Products, Camden, New Jersey, AD 460 381, June 1, 64 - Sept 30, 64.

Rice, R., Schauer, R. F., Terman, L. M., "High to Low Order Numeric Processing," IBM Research, Yorktown, New York, Jan 20, 1962.

Richards, Paul, "Parallel Programming," Technical Operations Research, Burlington, Mass., AF 33 (600) - 35190, BTP 62-0376, AD 261 623, Div 15/1 30/1, Aug 60.

Robbi, A. D., Ricci, R., "Transfluxor Content-Addressable Memory," Proc. of the International Conference on Nonlinear Magnetics, Apr. 64, p 8-3-1.

Roberts, M. deV., "A Programming Proposal for a Computer Design," IBM Research, Yorktown, New York, RC-794, IBM CONFIDENTIAL, Aug 29, 1962.

Roberts, M. deV., "Associative Memories and the One Level Storc," IBM Research, Yorktown, New York, RC-807, IBM CONFIDENTIAL, Sept 26, 1962.

Rogers, J. L., "Research on Cryogenic Associative Memories," Quarterly Progress Report for Period Ending 30 Nov 1962, Space Technology Labs, TP 64H1-4668, 16 Jan 1963.

Rogers, J. L., "Research on Cryogenic Associative Memories," Quarterly Progress Report for Period Ending 28 Feb 1963, Space Technology Labs, 11 Apr 1963.

Rogers, J. L., "Research on Cryogenic Associative Memories," Quarterly Progress Report for Period Ending 30 May 1963, Space Technology Labs, 8 Aug 1963.

Rogers, J. L., Wolinsky, A., "Associative Memories and Their Cryogenic Implementation," TRW Space Laboratories, Redondo Beach, Calif., 8670-6007-RU-000, DDC, Dec 63.

Rogers, J. L., Wolinsky, A., "Research on Cryogenic Associative Memories," Final Summary Report, 28 May 64, TRW Space Tech. Labs., NONR 3839(1001), 1 Jun 62 - 30 May 64.

Rosengerger, J., Lindquist, A. B., Seeber, R. R., "Cryogenics Memory Plane Interconnection Techniques," IBM Poughkeepsie, Poughkeepsie, New York, AD 622 819, DIV 30/2, 8/2, Oct 65.

Rosenblatt, F., "Perceptron Simulation Experiments," Proc. IRE, 1960, Vol. 48, pp 301-309.

Rosin, R. F., "An Organization of an Associative Cryogenic Computer," Proc. SJCC, San Francisco, Calif., May 62, pp 203-212.

Rosin, R. F., "List Structures and Their Implementation Through Advanced Machine Design," IBM Research, Yorktown, New York, RC-297, Aug 60.

Ross, D. T., "A Generalized Technique for Symbol Manipulation and Numerical Calculation," Comm. ACM, Vol. 4, No. 3, Mar. 1961.

Roth, J. P., "Systematic Design of Automata," IBM Research, Yorktown, New York, RC-1425, 16 Jun 65.

Rothman, S., "The RW-400 Data Processing System," Proc. Auto-Math Conference, International Congress of Information Processing, Paris, June 1959.

Rowe, A. J., Brock, P., "Use of Hybrid Computing in Design Automation," Hughes Aircraft, Rand Corp., AD 613 002, Mar 65.

Rowland, C. A., Berge, W. O., "A 300 Nanosecond Search Memory," Proc. FJCC, Las Vegas, Nev., Nov 1963.

Rybak, Franklyn M., "Study to Determine the Applicability of the Solomon Computer to Command and Control," Vol I. Information Storage, Retrieval and Communication System Control, Westinghouse Electric, Baltimore, Md., AD 454 765, Div 30/1 5/1, Oct. 64, 194 p.

Sagnis, J. C., Stuckert, P. E., "Cross Core Memory Element," IBM Research, Yorktown, New York, RC-767, IBM CONFIDENTIAL, Sept. 62.

Schauer, R. F., "Variable Field Length: Its Effect on the ADAM System Design," IBM Research, Yorktown, New York, RC-1138, 6 Mar 64.

Schauer, R. F., Meggitt, J. E., "Evaluating the Logical Design of the Problem Oriented Symbol Processor," Final Report, IBM Research, Yorktown, New York, AF 19(628)-3257, 1 Nov 63.

Schlaeppi, H. P., "A Formal Language for Describing Machine Logic, Timing and Sequence (LOTIS)," IBM Research, Yorktown, New York, RZ-125, 24 Dec 63.

Schlaeppi, H. P., "Notes on Microprogramming of Digital Systems," IBM Research, Zurich, Switzerland, RZ-33, July 1958.

Schoene, L. P., Jr., Murray, P. J., "The Application of List Processing Techniques to Intelligence Data Processing," Final Report, IBM-FSD, Rockville, Md., Task No. 0273, IBM CONFIDENTIAL, 31 Dec 62.

Seeber, R. R., Jr., Lindquist, A. B., "Associative Logic for Highly Parallel Systems," Proc FJCC, Las Vegas, Nev., Nov. 12-14, 1963, p. 489.

Seeber, R. R., Jr., Lindquist, A. B., "Associative Memory with Ordered Retrieval," IBM Journal of Research and Development, Vol. 6, No. 1, Jan. 1962, p. 126.

Seeber, R. R., Jr., Scriver, A. J., "Associative Self-Sorting Memory," Proc. EJCC, Vol. 18, Dec. 1960.

Seeber, R. R., Jr., "Associative Self-Sorting Memory Revised," IBM Data Systems, Poughkeepsie, New York, TR-00, 756, Nov. 60.

Seeber, R. R., Jr., "Cryogenic Associative Memory," National Conference of the ACM, Aug. 1960.

Seeber, R. R., Jr., Lindquist, A. B., "Mass Fabrication, Highly Parallel Systems, and Associative Logic," IBM, Poughkeepsie, New York, TIC 63AS 0518, IBM CONFIDENTIAL, 22 May 62.

Seeber, R. R., Hartman, F. B., "Memory and Circuits Therefor," U. S. Patent No. 3,121,217, Feb 64.

Seeber, R. R., "Symbol Manipulation With an Associative Memory Preprints," National Conference of the ACM, Sept 61.

Selfridge, O. G., "Pattern Recognition and Modern Computers," Proc. WJCC, 1955, pp 91-93.

Shahbender, R., et al, "Laminated Ferrite Memory," Proc. F.JCC, Las Vegas, Nev., Nov. 1963.

Shaw, J. C., et al, "A Command Structure for Complex Information Processing," Proc WJCC, 1958.

Shay, G., Jr., Spruth, W. G., "Analysis of a File Addressing Method."

Shooman, Wm., "Parallel Computing with Vertical Data," SDC, Santa Monica, Calif., AD 276 593, Jan 1961.

Shooman, W., "Parallel Computing With Vertical Data," Proc EJCC, New York, Dec 13-15, 1960.

Simmons, G. J., "Application of an Associatively Addressed Distributed Memory," Proc. SJCC, Apr 64, p. 493.

Simmons, G. J., "A Mathematical Model for an Associative Memory," Sandia Corp. Report, SCR-641, April 1963, Sandia Corp. Report, SCR-621, April 1963.

Singer, T., Schupp, P., "Associative Memory Computers from the Programming Point of View," Mitre Corp., ESD-TDR-63-245, AD 416 301, Aug 1963.

Slade, A. E., McMahon, H. O., "A Cryotron Catalog Memory System," Proc. EJCC, Dec 56, p 120.

Slade, A. E., Smallman, C. R., "Thin-Film Cryotron Catalog Memory," Automatic, Aug. 1960, p. 48.

Slade, A. E., "The Woven Cryotron Memory," Proc. International Symposium on the Theory of Switching, 1959, p. 326.

Slotnick, D. L., et al, "The Solomon Computer - A Preliminary Report," Workshop on Computer Organization, A. Barum and M. Knapp, Editors.

Slotnick, D. L., Borck, W. C., McReynolds, "The Solomon Computer," Proc. [illegible], San Francisco, Calif., May 1962.

Smith, E. . "A Directly Coupled Multiprocessing System," IBM Systems Journal, Sept/Dec 1963.

Smith, R. V., "A Programmed Associative Memory for Use in Compiling," SRI - Term Paper No. 2-40, IBM Corporate, New York City, IBM CONFIDENTIAL, 28 Apr 1961.

Smith, R. V., Senzig, D. N., "Computer Organization for Array Processing," IBM Research, Yorktown, New York, RC-1330, 9 Dec 64.

Space Technology Labs, "Feasibility Study for a Cryogenic Associative Memory," Report: Proposal 0759.00, July 1961.

Sperry-Rand Corp., "UNIVAC Search Memory," MO 5562, Sperry-Rand Corp., Univac Park, St. Paul, Minn.

Sproul, W. W., "Microprogram Computers with Minimum Redundancy Memories," IBM-FSD, Owego, New York, 63-533-014, Dec. 1963.

Stanford Research Institute, "Development of a Multiple Instantaneous Response File," The AN/GSQ-81 Document Data Indexing Set, AD 609 126, Stanford Research Institute, Menlo Park, Calif., Oct 1964.

Strachey, "Time Sharing in Large Fast Computers," Proc. IFIPS, 1959.

Tapscott, R. P., "An Algorithm for the Reduction of Intermediate Storage Utilization Costs," RC-599, IBM Research, Yorktown, New York.

Thompson, R. N., Wilkinson, J. A., "The D 25 Automatic Operating and Scheduling Program," Proc SJCC, 1963.

Toxen, A. M., Burns, M. J., "Critical Fields of Thin Superconducting Films, II Mean Free Path Effects in Indium-Tin Alloy Films," IBM Research, Yorktown, New York, Sept 10, 1962.

TRW, "Computer Associative Memory Study," TRW-Space Technology Labs, Redondo Beach, Calif., AD 442 950, Div 30/1, July 64.

TRW, "Computer Associative Memory Final Report," TRW Inc., DDC, TP 64K1-5764, Not for For. Dist., 15 Jul 64.

Turm, R., "Assignment of Inventory of a Variable Structure Computer," PhD Dissertation, U.C.L.A., Jan. 1963.

Tuttle, G. T., "How to Quiz a Whole Memory at Once," Electronics, 15 Nov 63, p 43.

Uhr, L., Vossler, C., "A Pattern Recognition Program that Generates, Evaluates and Adjusts Its Own Operators," WJCC, May 1961, pp 555-569.

Unger, S. H., "A Computer Oriented Toward Spatial Problems," Proc, IRE, Oct. 1958, pp 1744-1750.

Wagner, E. G., "An Approach to Modular Computers I. Spider Automata and Embedded Automata," IBM Research, Yorktown, New York, RC-1107. Jan. 1964.

Wagner, E. G., "Modular Computers II. Graph Theory and the Interconnection of Modules," IBM Research, Yorktown, New York, RC-1414, 4 Jun 65.

Wagner, E. G., McCarthy, J., "Tag Memory," U. S. Patent No. 3,093,814, Jun 63.

Wang, C. P., Ruehli, A. E., "A Transistor-Tunnel Diode Cell for Associative Memories and Multiple-Word Access Memories," 65C-001359-MF003, IBM CONFIDENTIAL.

Wanner, V. R., "The Logical Design of a Multi-Channel Device for the Retrieval of Information," ONR Report No. ACR-93, April 1964.

Webster, W. W., "Problems In Utilizing Tabular Languages, with Examples Drawn from 1401 Card Program Generator," IBM Corporate, New York City, Dec 1961.

Weinstein, H., "Proposals for Ordered Sequential Detection of Simultaneous Multiple Responses," IEEE Trans. on Electronic Computers, Oct. 63, p. 564.

Westinghouse Corp., "Study and Investigation to Develop Compiler Techniques Required for Programming the Parallel Network Computer," Westinghouse Defense and Space Center, Baltimore, Md., AD 602, Div 3 /1, Jun 64.

Westinghouse Corp., "Parallel Network Computer (Solomon)", Westinghouse, Baltimore, Md., AD 419 318, Div 30/1, 8/1, April 1963.

Westinghouse Corp., "Multiple Processing Techniques," Westinghouse, Baltimore, Md., AD 602 693, Div 30/1, June 64.

Westinghouse Corp., "Parallel Network Computers (Solomon) Solomon Breadboard Technical Report," Westinghouse Electric Corp., Air Arm Division, Baltimore, Md., AF 30(602)2724, AD 419318, 15 Apr. 1963.

Wigington, R. L., "A Machine Organization for a General Purpose List Processor," IEEE Transactions on Electronic Computers, Dec. 1963.

Winters, R. E., "Special Intelligence Processor," Progress Report, Jan-Dec 1962, IRAD 0270, IBM-FSD, May 1963.

Wiseman, N. E., "Applications of List Processing Methods to the Design of Interconnections for a Fast Logic System," Electrical Communication, ITT, Vol 38, No. 3, 1963.

Yang, C. C., Tou, J. T., "Systematic Design of Cryogenic Logic Circuits," Computer Sciences Lab, AD 617 501, Div 30/2 25/6, 1964.

Yngue, V. H., COMIT, Comm ACM, Mar 1963.

Younker, E. L., Heckler, C. H., Jr., Masher, D. P., Yarborough, J. M., "Design of an Experimental Multiple Instantaneous Response File," Proc. SJCC, Apr. 64, p 515.

Younker, E. L., Heckler, C. H., Jr., Masher, D. P., Yarborough, J. M., "Development of a Multiple Instantaneous Response File: the AN/GSQ-81 Document Data Indexing Set," Stanford Research Institute, Menlo Park, California, AD 609 126, Div 32/1, 30/1, Oct 64.

Zucker, M. S., Griffin, J. F., Haimes, M. J., Patton, S. K., "Control Systems Technology," IBM Advanced Systems, Endicott, New York, TR-17-058-RD, Dec 61.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Federal Systems Div IBM Corp 7220 Wisconsin Ave, Bethesda, Md 20014	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP N/A

3. REPORT TITLE

Advanced Computer Organization

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Final Report

5. AUTHOR(S) (Last name, first name, initial)

Baker, F.T.	Schenken, J.D.
Forbes, C.H.	Triest, W.E.
Jacobs, N.	Walker, T.P., Jr.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
~~February 1966~~ May 1966	302	293

8a. CONTRACT OR GRANT NO. AF30(602)-3573	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. 4594	
c. Task # 459406	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) RADC-TR-66-148
d.	

10. AVAILABILITY/LIMITATION NOTICES: This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMLI), GAFB, N.Y. 13440.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center Griffiss AFB, New York 13440

13. ABSTRACT

This study resulted in a design of an advanced general-purpose computer, including its functional organisation and programming. The design is based on content-addressable parallel search memories and the computer has parallel processing capability. It resulted from investigations in several important areas of non-numeric processing and symbol manipulation, and the design studies which were carried out in each area. In addition to the general-purpose computer and the individual design studies, a number of associative processing techniques were developed for use with such equipment.

DD FORM 1 JAN 64 1473

UNCLASSIFIED

Security Classification

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Associative Processor						
Associative Memory						
Content-Addressable Memory						
Parallel Search Memory						
Non-Numeric Processing						
Symbol Manipulation						
Data Extraction						
Indexing						
Word Frequency						
Formatted Files						
Pattern Recognition						
Textual Error Correction						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization (*corporate author*) issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers (*either by the originator or by the sponsor*), also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through ____________________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through ____________________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through ____________________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring (*paying for*) the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.

UNCLASSIFIED

Security Classification